Drucksachenkategorie

Dynamic Modelling of Robots with Kinematic Loops

Ruf Oliver

DLR

RMC, Institut für Robotik und Mechatronik	BJ.: 2014
	IB. Nr.: 572-2014/33

MASTERARBEIT

DYNAMIC MODELLING OF ROBOTS WITH KINEMATIC LOOPS

Freigabe:	Der Bearbeiter:	Unterschriften
	Oliver Ruf	
	Betreuer:	
	Roberto Lampariello	
	Der Institutsdirektor	
	Prof. Alin Albu-Schäffer	

Dieser Bericht enthält 75 Seiten, 26 Abbildungen und 30 Tabellen

Ort: Oberpfaffenhofen	Datum:	Bearbeiter:	Zeichen:

Oliver Ruf

Dynamic Modeling of Robots with Kinematic Loops

School of Electrical Engineering

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 1.8.2014

Thesis supervisors:

Prof. em. D. Sc. (Tech.) Aarne Halme
Aalto University, Helsinki

Prof. Thomas Gustafsson
Luleå University of Technology, Kiruna

Thesis advisors:

Roberto Lampariello
DLR German Aerospace Center, Oberpfaffenhofen

Martin Stelzer
DLR German Aerospace Center, Oberpfaffenhofen

Aalto University
School of Electrical Engineering

abstract of the
master's thesis

Author: Oliver Ruf

Title: Dynamic Modeling of Robots with Kinematic Loops

Date: 1.8.2014 Language: English Number of pages: 10 + 75

Department of Automation and Systems Technology

Professorship: Automation Technology Code: AS-84

Supervisors: Prof. em. D. Sc. (Tech.) Aarne Halme (Aalto)
Prof. Thomas Gustafsson (LTU)

Advisors: Roberto Lampariello (DLR)
Martin Stelzer (DLR)

Closed kinematic loops occur in many different robotic systems. Simply grasping an object with two or more arms, generates a closed kinematic loop. Recently, ESA commissioned a feasibility study for the deorbiting of Europe's largest earth observing satellite – Envisat. It is expected that multiple arms are needed in order to grasp this satellite, which would generate a closed kinematic loop. The German Aerospace Center DLR would like to contribute to this feasibility study and simulate the deorbiting of Envisat. The SpaceDynamics Library, a simulation environment of the DLR, is however incapable to simulate robots with closed kinematic loops. The thesis discusses the dynamic modeling of robots with closed kinematic loops and shows the implementation and validation of an algorithm, in order to enable the dynamic modeling of closed loops with the SpaceDynamics Library.

Keywords: Dynamic Modeling, Kinematic Loops, Closed Loops, Space Robots

Contents

List of Symbols and Abbreviations

General Symbols

$\boldsymbol{a}_i$	acceleration of link i in absolute coordinates
$\boldsymbol{C}$	matrix of Coriolis and centrifugal terms
$\boldsymbol{f}_i^*$	force that acts on link i, depending on spatial inertia
$\boldsymbol{f}_i^{\boldsymbol{x}}$	external force that acts on link i
$\boldsymbol{f}_i$	total force that is transmitted through joint i (from link $i-1$ to link i)
$\boldsymbol{g}$	gravity vector
$\boldsymbol{H}$	joint space inertia matrix
$\boldsymbol{I}_i$	spatial Inertia of link i in absolute coordinates
$\boldsymbol{I}_i'$	spatial Inertia of link i in link coordinate system i
n	number of links and joints in the system
q_i	joint position or displacement of joint i
$\boldsymbol{q}$	vector of joint positions
$\dot{q}_i$	joint velocity of joint i
$\dot{\boldsymbol{q}}$	vector of joint velocities
$\ddot{q}_i$	joint acceleration of joint i
$\ddot{\boldsymbol{q}}$	vector of joint accelerations
$\boldsymbol{s}_i$	vector of link i in absolute coordinates
$\boldsymbol{s}_i'$	vector of link i in link coordinate system i
τ_i	force of joint i
$\boldsymbol{\tau}$	vector of joint forces
$\boldsymbol{v}_i$	velocity of link i in absolute coordinates
${}_i\boldsymbol{X}_j$	homogeneous transformation between coordinate system i and j

Note: Bold symbols represent vectors, whereas normal symbols represent scalars.

Additional Symbols of Chapter 4

$\boldsymbol{J}_C$	constraint matrix of all closed Loops
$\boldsymbol{J}_{Cm}$	linearly independent rows of $\boldsymbol{J}_C$
$\boldsymbol{J}_G$	linearly dependent columns of $\boldsymbol{J}_{Cm}$
$\boldsymbol{J}_{Li}$	velocity constraint matrix of Link L
$\boldsymbol{J}_S$	linearly independent columns of $\boldsymbol{J}_{Cm}$
m	number of independent constraints
N_A	number of actuated joints
N_F	number of degrees of freedom
N_J	number of all joints in the closed loop system
N_O	number of joints in the open loop system
$\boldsymbol{S}$	Jacobian matrix
$\boldsymbol{\tau}_A$	actuator forces in the closed loop system
$\boldsymbol{\tau}_G$	generalized forces
$\boldsymbol{\tau}_O$	joint forces in the open loop system
$\boldsymbol{\theta}_A$	joint angles of actuated joints
$\boldsymbol{\theta}_G$	generalized coordinates (independent joints)
$\boldsymbol{\theta}_J$	all joint angles
$\boldsymbol{\theta}_O$	joint angles of the open loop system
$\boldsymbol{\theta}_S$	joint angles of the dependent joints
$\boldsymbol{W}$	Jacobian matrix

Abbreviations

DEOS	Deutsche Orbitale Servicing Mission
DLR	German Aerospace Center
EVA	Extra-vehicular activity
ESA	European Space Agency
GEO	Geostationary orbit (altitude 35786km)
IADC	Inter-Agency Space Debris Coordination Committee
LEO	low Earth orbit (altitude 160km - 2000km)
RMC	Robotic and Mechatronic Center
SDL	SpaceDynamics Library
TORO	**To**rque controlled humanoid **Ro**bot of the DLR

List of Figures

List of Tables

Listings

1. Introduction

Artificial objects exist in the space environment of Earth since the launch of the first satellite Sputnik-1 in 1957. Once in earths orbit, artificial objects remain there for decades. In the low-earth-orbit (LEO), atmospheric drag and solar pressure effects lead to altitude loss of the objects until they finally burn up in the Earth's atmosphere. According to [1], only 6% of artificial objects in space were active satellites in 2002.

All man-made objects orbiting the earth, which are not functional, are called space debris [2] and make up the biggest part of the space object population. Space debris consists of inactive satellites, parts of launch vehicles and products of on-orbit collisions and explosions. The latter is the largest contributor to space debris and therefore the most dangerous. The debris population in operationally important orbits is growing due to new space missions. A chinese anti-satellite test in 2007 and the collision of the Iridium 33 and the Cosmos 2251 satellite in 2009 led to a large growing of debris in recent years [3]. Every artificial object can collide with other objects in space and produce even more debris, which is known as the cascading effect. Space debris represents a huge hazard to active and future satellites because of possible collisions. To attack the task of reducing space debris, the Inter-Agency Space Debris Coordination Committee (IADC) was formed in 1993. In 2002 the members agreed on the guidelines to decrease space debris.

One possible solution to reduce space debris is the servicing of satellites. Servicing includes life extension, repair and removal of satellites [4]. Especially in the geostationary orbit (GEO) and in polar orbits the removal of satellites is needed. In these orbits almost no atmospheric drag effects exist, that would lead to a self regulated removal of the debris. In addition, these orbits are operationally important for Earth observation and communication. One example of on-orbit servicing is DEOS (Deutsche Orbitale Servicing Mission) [5]. The goal of the DEOS mission is to grasp a tumbling target satellite with the manipulator of a service satellite and to eventually deorbit the target satellite.

The largest earth-observing satellite of the European Space Agency (ESA) is Envisat. It is an **Envi**ronmental **sat**ellite with a launch mass of 8221 kg which is orbiting the Earth on a sun-synchronous polar orbit at an altitude of 790 km. After 10 years of operation, the contact to Envisat was lost on April 12$^{\text{th}}$, 2012. Since then, Envisat is not controllable anymore and is therefore considered as space debris. At the 63$^{\text{rd}}$ International Astronautical Congress, Martha Mejia-Kaiser described Envisat as a large danger for other satellites [6]. Due to little drag effects at the current altitude, Envisat will remain in the orbit of Earth for around 150 years. With the cross-section of 26 meters the probability of a collision with another satellite is

very high. Against the guidelines of the IADC, Envisat was operated until too little fuel was left for moving it to a lower orbit where a shorter lifetime would result. For this reason ESA could be held liable for occurring collisions. A feasibility study for the deorbiting of Envisat was now commissioned by ESA. Considering the size and mass, it is expected that multiple arms are necessary to grasp Envisat for deorbiting. By grasping Envisat with multiple arms, a closed kinematic loop would occur. This thesis explains now why the *Dynamic Modeling of Robots with Kinematic Loops* is needed for the simulation of the deorbiting of Envisat and how the modeling can be done.

1.1. Kinematic and Dynamic Modeling of Robots

1.1.1. Kinematic Modeling

The kinematic modeling of robots sets the position of the end effector of a robot in relation to the positions and values of the robots joints. This is done without considering the forces or torques that are responsible for the motion [7]. The kinematic modeling plays an important role in the design and analysis of a robot and is the foundation of the dynamic modeling.

Forward kinematics is the computation of the pose (position and orientation) of the end effector of a robot for given joint positions and values and the geometric structure of the robot. If the velocity of all joints is added to the given values, the resulting velocity of the end effector can be calculated. This is called forward instantaneous kinematics.

Inverse kinematics is exactly the opposite: It is the computation of the joint position and values required for a given pose of the end effector. Similar to the forward kinematics, the computation of the joint velocities for a given end effector velocity is called inverse instantaneous kinematics.

1.1.2. Dynamic Modeling

In contrast to the kinematic modeling, the dynamic modeling of robots sets the forces of actuators and forces due to contact with external objects in relation to the acceleration and motion of the robot [8]. Therefore the dynamic modeling, especially the *inverse* and the *forward dynamics* are an important part in simulating or controlling robotic systems.

The inverse dynamics is the computation of the required actuator torques and forces for a given trajectory the robot should perform. The trajectory of the robot is described by the joint angles, the joint velocities and the joint accelerations. To control a robotic system, the actuator torques and forces that are required for the robot to perform a certain motion have to be calculated. Therefore the inverse dynamics is the computation that is needed for controlling a robotic system.

In contrast to the inverse dynamics, the forward dynamics is the computation of the joint accelerations for given actuator torques and forces. In this case the trajectory performed by the robot is calculated for certain torques and forces that are applied to the actuators. This computation is needed for simulating the motion of the robotic mechanism.

1.2. Kinematic Loops and Kinematic Trees

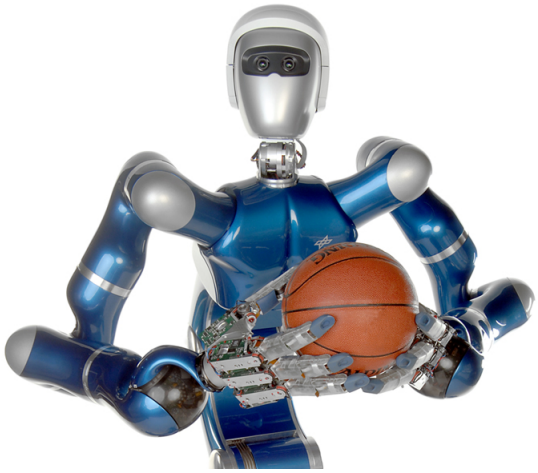


Figure 1.1.: Humanoid Justin holding a basketball, DLR

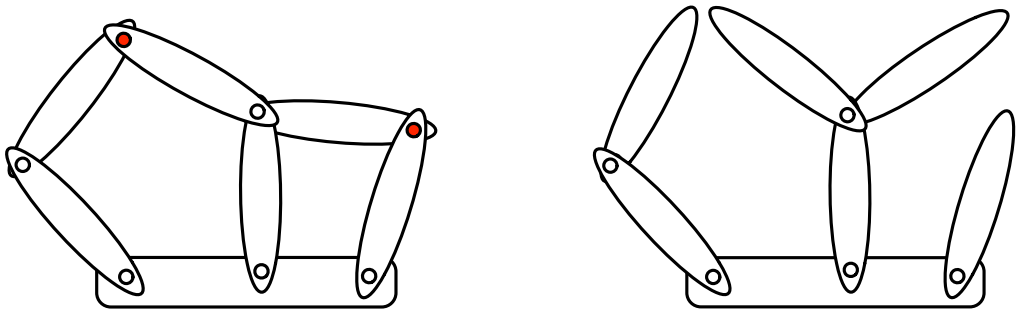
1.2.1. Kinematic Loops

Closed loops or kinematic loops occur in many different robot mechanisms. According to Featherstone [9, p. 155] a "kinematic loop exists in a mechanism if it is possible to trace a circuit from some link back to itself without traversing any joint more than once". This definition holds true even if there is an external object involved, as for example a robot carrying a rigid object with two robotic arms. In Figure 1.1 the humanoid Justin from the German Aerospace Center (DLR) forms a closed loop with its two arms by holding a basketball. More examples of robotic systems with kinematic loops are (space-) robots with cooperative manipulators, humanoid robots or parallel robots and will be described in chapter 2.

1.2.2. Kinematic Trees

A mechanism that contains however no kinematic loops is called tree-structure open kinematic chain, open loop system or kinematic tree. According to Featherstones definition of kinematic loops, in a kinematic tree it is only possible to trace a circuit from a link back to itself by traversing some joints more than once. The difference between kinematic loops and kinematic trees is shown in Figure 1.2. The two

kinematic loops in Figure 1.2 (a) are cut at the red marked joints to form a kinematic tree structure shown in Figure 1.2 (b).



(a) Kinematic Loop System (b) Kinematic Tree System

Figure 1.2.: Comparison between Kinematic Loops and Kinematic Trees

1.3. Fixed Base, Free Flying and Free Floating Systems

Ongoing research regarding kinematic loops includes the dynamic modeling of robotic systems in space, where special dynamic effects occur [5]. These are for example closed loops in free flying and free floating robotic systems which represent particularities in space robotics. In fixed base robotic systems, that mostly occur on Earth, the movement of the robotic arms does not affect the pose of the base of the robot. It is assumed to be fixed. In contrast to fixed base robots, the base of a free flying or free floating robot changes its position and orientation according to the movement of the robots arm [10]. This is due to the conservation of momentum and is mostly the case for space robots. The difference between free flying and free floating base is that a free floating base is not actively controlled whereas the pose of a free flying robot is controlled by thrusters and reaction wheels. So far there has not been a free flying or free floating base robot with closed kinematic loops in space.

1.4. Motivation

The SpaceDynamics Library (SDL) [11] is an existing simulation environment at the Robotic and Mechatronic Center (RMC) of the DLR. It is used for calculating the kinematics and dynamics for all kind of robotic applications, especially free-floating orbital robots. The SpaceDyn library however lacks the functionality of modeling robotic systems with closed kinematic loops. The dynamics computation of a robotic system with closed kinematic loops is needed at the RMC in order to simulate a free-flying orbital robot with multiple arms that forms a kinematic loop while grasping an object. This simulation could be used for the feasibility study commissioned by ESA for the deorbiting of Envisat.

Different solutions to calculate the kinematics and dynamics of closed kinematic loops exist already in the literature, for example [9, 12–14]. One of the existing approaches should be selected and implemented in the SpaceDynamics Library to enable the modeling of closed kinematic loops. The chosen implementation should involve a state-of-the-art method in terms of computation time. The question that is answered with this thesis is: How to compute the inverse and forward dynamics for a fixed base, free-flying or free-floating robotic system including kinematic and actuation redundancy with N arms and L closed loops.

The goal of the research is the simulation of the kinematics and dynamics of a closed loop robotic system with the SpaceDynamics Library. For successful completion of the thesis, first one existing solution for the kinematic and dynamic modeling of robots with closed kinematic loops has to be chosen. Second, the chosen algorithm has to be implemented in the SpaceDynamics Library. Finally its functionality has to be validated with SIMPACK, that is a commercial multi-body simulation software, in order to allow the simulation of a use case scenario with the SpaceDynamics Library.

A possible use case scenario for the SpaceDynamics Library is the Envisat case, where the movement of a satellite that is grasped with multiple arms should be simulated. The Robonaut case (see section 2.3) is a possible scenario as well. In this case, on-orbit manipulation and assembly tasks with multiple arms are performed. The SpaceDynamics Library is however not restricted to space robot modeling, therefore another use case scenario could be the biped walking case. For this case, the simulation of a humanoid robot forming a closed loop with two legs on the floor is needed.

A challenging issue is to understand the theory, which is quite complex. Since the software should be able to control a robotic system, the dynamics computation has to be done in real time. Therefore the implementation of the software is also challenging. Furthermore the validation of the algorithms has to be done in a very accurate way, to confirm the functionality and validity of the computed dynamics.

1.5. Outline

After this introduction, chapter 2 shows now some examples and applications of robot mechanisms including closed kinematic loops in different environments. After that, the general dynamic modeling of robots is explained in chapter 3. Chapter 4 extends the general dynamic modeling by describing methods for systems with kinematic loops. Within this scope, the report presents two different approaches of Featherstone [9] and Nakamura [12] to compute the inverse and the forward dynamics of closed loops. Chapter 5 describes how the the SpaceDynamics Library works and gives detailed information about important functions. In chapter 6 the implementation of the algorithms to calculate the dynamics of closed loop systems is shown. Furthermore, the tests and validation of the implemented algorithms is presented. Chapter 7 shows the results of this work and gives a summary of the report. Finally, an outlook on further work is given.

2. Examples and Applications of Kinematic Loops

This chapter gives a summary of the most important examples and applications of robots with kinematic loops. Among them are cooperative manipulators, humanoid robots, space robots and parallel robots. These systems are described in detail and an overview of existing literature for further reading is given.

2.1. Cooperative Manipulators

Closed loops occur especially in cooperative manipulators and humanoid robots. Cooperative manipulation is defined as the manipulation of one object with multiple robotic arms [15]. Cooperative manipulation is used for certain tasks, that are not possible to perform with only one arm, for example unscrew the cap of a bottle. For this and other applications a second arm is needed. Furthermore, the ability to manipulate heavier objects with multiple arms than possible with only one arm is a goal of cooperative manipulators.

Figure 2.1.: Piano Mover's Problem, [16]

Figure 2.1 shows a piano that is moved by three cooperative mobile manipulators, where three closed kinematic loops occur. The motion planning for systems with kinematic loops is however more difficult than for kinematic tree systems. Therefore, Cortés [16] describes computational efficient motion planning algorithms for systems with closed loops.

2.2. Humanoid Robots

Figure 2.2.: Humanoid TORO, DLR

Humanoid robots are designed to perform the same manipulation tasks as humans [17]. Simply holding an object with two or more arms forms a kinematic loop [12]. A humanoid robot with two legs, standing on the floor is also creating a closed loop through the ground. Here the dynamic modeling is very important for the robot in order to not fall down due to different weight distribution on the legs. Figure 2.2 shows TORO, the **To**rque controlled humanoid **Ro**bot of the DLR, that is able to walk and to compensate hits against the legs. TORO is forming a closed loop with both legs on the floor.

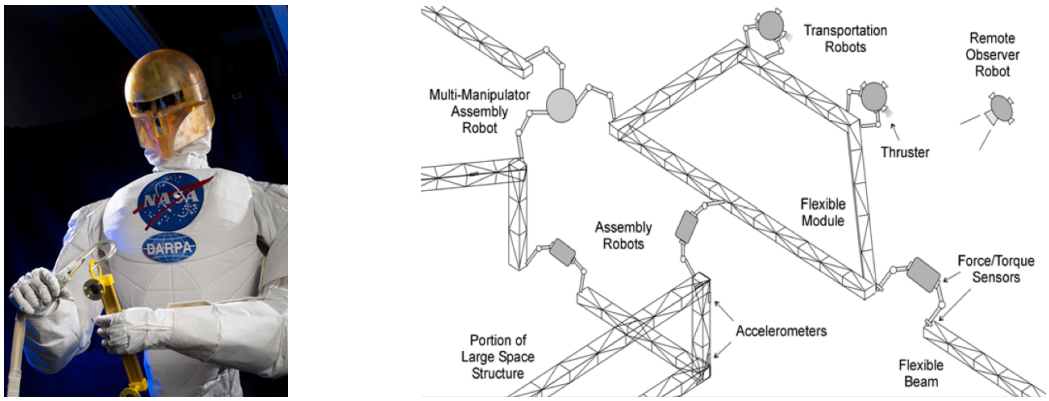
Much literature exists about the dynamic modeling of humanoid robots. [12] presents a general dynamic modeling method that is intended for the modeling of humanoids. Despite that, it describes the modeling of free flying robots with closed loops and is therefore very interesting. Due to the general approach this method is presented in chapter 4.

2.3. Space Robots with Multiple Manipulators

For the simulation of the deorbiting of Envisat, the dynamic modeling of robots with kinematic loops is needed. Thus, space robots are the main applications intended for the dynamics computation of this research. Although much literature exists to space robots with multiple manipulators, no free-floating or free-flying robot performing closed loop manipulation existed in space so far.

A generalized formulation of the dynamic equations for a space robot with open and closed chain configurations is described by [13]. Further the control and the kinematic and dynamic modeling of space robots with kinematic loops is described in [14, 18–20].

Future orbital robotic concepts include the control of autonomous space robots for construction [21] and on-orbit maneuvering of large space structures [22]. Future exploration missions will also make use of in-space assembly [23], where closed loops occur as well.



(a) Robonaut, [10]

(b) Construction of Large Space Structures, [21]

Figure 2.3.: Space Robots

Another future concept of space robotics is the Robonaut [10] developed by NASA. It is a humanoid space robot intended to perform extra-vehicular activity (EVA) tasks and planetary exploration. The controlling is done by Telepresence, which is similar to a remote control. The astronauts are able to steer the Robonaut by joysticks. The Robonaut is able to perform closed-loop manipulation, but it is considered to be always connected to a spacecraft and is therefore a fixed-base system. Of course the movement of the Robonaut adds a dynamic momentum on the pose of the spacecraft, which is a free flying base system. In case of the ISS beeing the spacecraft, this force is very small due to the mass differences and can therefore be neglected.

2.4. Parallel Robots

In parallel robots, closed-loops exist by definition: A parallel manipulator is a closed loop mechanism with one end effector that is connected to the base by multiple independent kinematic chains [24].

Figure 2.4 shows a Gough platform which is a well known example for a parallel robot. In that sense the term Hexapod describes a parallel robot with six independent kinematic chains (legs). The major advantage of parallel robots is the ability to handle big loads compared to serial manipulators. Another advantage is the high accuracy of the positioning of the end effector. Since the end effectors position is dependent on multiple kinematic chains, the workspace of a parallel robot is rather small.

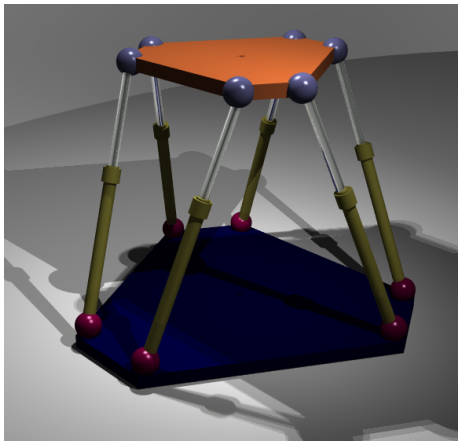


Figure 2.4.: Hexapod Platform [1]

The main applications for parallel robots are flight and driving simulators, high speed and high accuracy manufacturing and mirror alignment system in telescopes. An example of high speed and high accuracy manufacturing is PCB assembly. In February 1999, a fixed base parallel robot with 8 legs was used in space during the Space Shuttle mission STS-63, to prevent the payload from vibration [24].

The dynamic modeling of parallel robots is described in [25], where Rose presents a method to compute the inverse dynamics in real time for controlling the parallel robot Triglide. However the dynamic modeling was assumed to not fit to the general approach of this research, therefore this method is not described in detail.

[1] http://en.wikipedia.org/wiki/File:Hexapod0a.png, (March 10, 2014)

3. Dynamic Modeling of Kinematic Trees

As already described in the introduction, the dynamic modeling represents the relationship between the motion of a robot and the forces that act on a robot. These forces are the forces and torques that are applied to the actuators of the robot as well as the external forces that result due to the contact of the robot with other objects. If the actuator forces $\boldsymbol{\tau}$ and the contact forces $\boldsymbol{f}$ are known, the resulting joint accelerations $\ddot{\boldsymbol{q}}$ are calculated by the forward dynamics. This calculation is needed for simulating robotic systems.

The motion of of the robot is described by the the joint angles $\boldsymbol{q}$, the joint velocities $\dot{\boldsymbol{q}}$ and the joint accelerations $\ddot{\boldsymbol{q}}$. If the motion of a robot is known, the required actuator forces $\boldsymbol{\tau}$ in order to perform this motion are calculated by the inverse dynamics. Therefore, the inverse dynamics is needed for the controlling of a robotic system.

This chapter explains now the basic equations for the dynamic modeling of kinematic tree systems and shows the most efficient way to perform the inverse dynamics calculation, which is needed for the dynamic modeling of robots with closed loops.

3.1. Equations of Motion

The equations of motion describe the relationship between the actuator forces and the motion of a robot. Therefore the equations of motion are the fundamental equations of the dynamic modeling. The joint-space formulation of the equations of motion according to [8] is

$$\boldsymbol{H}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{g}(\boldsymbol{q}) = \boldsymbol{\tau} \tag{3.1.1}$$

The vectors $\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}, \boldsymbol{g}, \boldsymbol{\tau} \in \mathbf{R}^{N_F}$ represent the joint position $\boldsymbol{q}$, the joint velocity $\dot{\boldsymbol{q}}$, the joint acceleration $\ddot{\boldsymbol{q}}$, the gravitational terms $\boldsymbol{g}$ and the forces $\boldsymbol{\tau}$, with the degrees of freedom of the system N_F. $\boldsymbol{H} \in \boldsymbol{R}^{N_F \times N_F}$ is the generalized inertia matrix of the system. From $\boldsymbol{C} \in \boldsymbol{R}^{N_F \times N_F}$, the Coriolis and centrifugal terms are represented by $\boldsymbol{C}\dot{\boldsymbol{q}}$. If a force $\boldsymbol{f}$ is applied to the end-effector of the robot, the term $\boldsymbol{J}^T\boldsymbol{f}$ has to be added to the right side of 3.1.1, where $\boldsymbol{J}$ is the Jacobian matrix of the end-effector. The dependencies are shown by the brackets: The inertia matrix $\boldsymbol{H}$ depends on $\boldsymbol{q}$, $\boldsymbol{g}$ depends on $\boldsymbol{q}$ and $\boldsymbol{C}$ depends on $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$.

The equations of motion 3.1.1 can be used to calculate the inverse dynamics of the system by calculating $\boldsymbol{\tau}$ directly. By solving the equation of motion for $\ddot{\boldsymbol{q}}$, it can be used to calculate the forward dynamics:

$$\ddot{\boldsymbol{q}} = \boldsymbol{H}(\boldsymbol{q})^{-1} \left[\boldsymbol{\tau} - \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} - \boldsymbol{g}(\boldsymbol{q})\right] \tag{3.1.2}$$

However, the computational complexity of computing the inverse and forward dynamics directly from these equations is typically $O(n^4)$ [9] and therefore very high. This is mostly due to the calculation of $\boldsymbol{H}$ with complexity $O(n^2)$ and $\boldsymbol{C}$ with complexity $O(n^3)$. The next section introduces the recursive Newton-Euler algorithm, that reduces the computational complexity of the inverse dynamics to $O(n)$.

3.2. Inverse Dynamics - Recursive Newton-Euler Method

Inverse dynamics is the computation of required actuator torques $\boldsymbol{\tau}$ of a robot in order to perform a given trajectory. The actuator torques are needed to control a robotic system. Therefore, the inverse dynamics are part of the control loop. To allow real-time control, the calculation of the inverse dynamics must be executed very fast.

The most efficient way to calculate the inverse dynamics is the recursive Newton-Euler Method as described in [9]. It was first presented by Luh, Walker and Paul [26] and is based on recurrence relations, which reduce the computational complexity from $O(n^4)$ to $O(n)$.

3.2.1. Recurrence Relations

In general, a recurrence relation is a recursive sequence, that defines each term as a function of the preceding terms. For example the Fibonacci numbers are defined by the following recurrence relation: $F_n = F_{n-1} + F_{n-2}$ with the starting values $F_0 = 0$ and $F_1 = 1$. A recurrence relation in dynamic modeling means for example, that the acceleration of link i is a function of the acceleration of link $i-1$. With the use of partial results that recurrence relations offer, unnecessary calculations are reduced. Therefore, algorithms that make use of recurrence relations are faster than algorithms without recursion. A more detailed description how to increase the efficiency of algorithms with the help of recurrence relations is given in [9, pp. 68]

3.2.2. The System Model

The use of recurrence relations requires a certain system model of the robot. Therefore a robot should be defined by n links, that are connected by n joints, each with one degree of freedom. Joint i connects link $i-1$ and link i, where link 0 is the base of the

robot. One coordinate system is attached to each link using the Denavit-Hartenberg scheme [27], so that coordinate system i is attached to link i. The advantage of the Denavit Hartenberg scheme is that instead of six, only four parameters are needed to describe the transformation from one link to an other. The four parameters are the link length, link twist, joint offset and joint angle. The spatial inertia $\boldsymbol{I}_i'$ of link i is constant in the coordinate system i. The possible motion of link i through joint i is described by $\boldsymbol{s}_i'$ in the coordinate system i. Both, $\boldsymbol{I}_i'$ and $\boldsymbol{s}_i'$ are expressed in i-coordinates.

${}_i\boldsymbol{X}_{i-1}$ is the coordinate transformation between coordinate system i and $i-1$. ${}_0\boldsymbol{X}_i$ describes the transformation between the link coordinate system i and the absolute coordinates. It is calculated by the recurrence relation

$$ {}_0\boldsymbol{X}_i = {}_0\boldsymbol{X}_{i-1}\ {}_{i-1}\boldsymbol{X}_i. $$

With this transformation all quantities expressed in link coordinates can be transformed into absolute coordinates, as for example:

$$ \boldsymbol{I}_i = {}_0\boldsymbol{X}_i\boldsymbol{I}_i' \text{ and } \boldsymbol{s}_i = {}_0\boldsymbol{X}_i\boldsymbol{s}_i', $$

where $\boldsymbol{I}_i$ and $\boldsymbol{s}_i$ is the spatial Inertia and motion of link i expressed in absolute coordinates.

To complete the system model, the system variables are defined. They describe the joint displacement q_i, the joint velocity $\dot{q}_i$, the joint acceleration $\ddot{q}_i$ and the joint force τ_i, each for joint i. If joint i is a rotational joint, q_i describes the joint angle, $\dot{q}_i$ the angular velocity, $\ddot{q}_i$ the angular acceleration, and τ_i the joint torque. All system variables for each link are merged in $\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}, \boldsymbol{\tau} \in \mathbf{R}^n$. For all applications, $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$ are considered to be known. If the joint accelerations $\ddot{\boldsymbol{q}}$ are known, then the joint forces $\boldsymbol{\tau}$ can be calculated by the inverse dynamics. In contrast to that, if the joint forces $\boldsymbol{\tau}$ are known, the joint accelerations $\ddot{\boldsymbol{q}}$ can be calculated by the forward dynamics.

3.2.3. The Recursive Newton-Euler Method

The Recursive Newton-Euler method calculates the inverse dynamics for a robot in three steps:

1. The velocity $\boldsymbol{v}_i$ and the acceleration $\boldsymbol{a}_i$ of each link is calculated.
2. The force $\boldsymbol{f}_i^*$, that acts on each link due to the motion and inertia of the link is calculated.
3. The joint forces $\boldsymbol{\tau}_i$ that generate the forces $\boldsymbol{f}_i^*$ in step 2 are calculated.

1. Velocity and Acceleration

The absolute velocity $\boldsymbol{v}_i$ of link i is depending on the velocity of the links' predecessor and the velocity across joint i. It is calculated by the recurrence relation

$$ \boldsymbol{v}_i = \boldsymbol{v}_{i-1} + \boldsymbol{s}_i\dot{q}_i \tag{3.2.1} $$

with the velocity of the base $\boldsymbol{v}_0 = 0$ for a fixed base robot. Differentiating Equation 3.2.1 gives the absolute acceleration $\boldsymbol{a}_i$ of link i:

$$\boldsymbol{a}_i = \boldsymbol{a}_{i-1} + \boldsymbol{v}_i \times \boldsymbol{s}_i \dot{q}_i + \boldsymbol{s}_i \ddot{q}_i \tag{3.2.2}$$

with the base acceleration for a robot on earth $\boldsymbol{a}_0 = -\boldsymbol{g}$, where $\boldsymbol{g}$ is the gravitational vector.

2. Link Force

The force that acts on link i is depending on the spatial inertia of the link:

$$\boldsymbol{f}_i^* = \frac{d}{dt}(\boldsymbol{I}_i \boldsymbol{v}_i) = \boldsymbol{I}_i \boldsymbol{a}_i + \boldsymbol{v}_i \times \boldsymbol{I}_i \boldsymbol{v}_i \tag{3.2.3}$$

The link velocities, accelerations and forces are calculated according to Equations 3.2.1 - 3.2.3 for each link, starting at the base ($i = 0$).

3. Joint Force

The spatial joint force that is transmitted from link $i-1$ to link i through joint i is calculated by the recurrence relation

$$\boldsymbol{f}_i = \boldsymbol{f}_{i+1} + \boldsymbol{f}_i^* - \boldsymbol{f}_i^{\boldsymbol{x}} \tag{3.2.4}$$

with the link force $\boldsymbol{f}_i^*$ calculated in step 2 and the external force $\boldsymbol{f}_i^{\boldsymbol{x}}$, which is acting on link i. This recurrence relation computes $\boldsymbol{f}_i$ for each link, starting at the end-effector ($i = n$) with $\boldsymbol{f}_n = \boldsymbol{f}_n^*$. The components of the spatial joint force, that act in the same direction as the joint motion form the joint forces $\boldsymbol{\tau}_i$ and are calculated for each joint by

$$\boldsymbol{\tau}_i = \boldsymbol{s}_i^T \boldsymbol{f}_i \tag{3.2.5}$$

This is the complete Recursive Newton-Euler Method for the inverse dynamics in absolute coordinates and has a computational complexity of $O(n)$.

3.2.4. Equations in Link Coordinates

For a more efficient algorithm, the calculations for each link i should be done in the link associated coordinate system i, since the transformation of vectors $\boldsymbol{v}, \boldsymbol{a}$ and $\boldsymbol{f}$ from one coordinate system to an other, is faster than the transformation of $\boldsymbol{s}'$ and $\boldsymbol{I}'$. The resulting equations for the link velocity, acceleration and force are then

$$\boldsymbol{v}_i = {}_i\boldsymbol{X}_{i-1} \boldsymbol{v}_{i-1} + \boldsymbol{s}_i' \dot{q}_i \tag{3.2.6}$$

$$\boldsymbol{a}_i = {}_i\boldsymbol{X}_{i-1} \boldsymbol{a}_{i-1} + \boldsymbol{v}_i \times \boldsymbol{s}_i' \dot{q}_i + \boldsymbol{s}_i' \ddot{q}_i \tag{3.2.7}$$

$$\boldsymbol{f}_i^* = \boldsymbol{I}_i' \boldsymbol{a}_i + \boldsymbol{v}_i \times \boldsymbol{I}_i' \boldsymbol{v}_i \tag{3.2.8}$$

Again the calculation of these equations is performed for each link, starting at the base ($i = 0$) with the starting values for the base velocity $\boldsymbol{v}_0 = 0$ for a fixed base robot and the base acceleration $\boldsymbol{a}_0 = -\boldsymbol{g}$ for a robot on earth.

The joint forces $\boldsymbol{\tau}$ are then calculated by

$$\boldsymbol{f}_i = {}_i\boldsymbol{X}_{i+1}\boldsymbol{f}_{i+1} + \boldsymbol{f}_i^* - \boldsymbol{f}_i^{\boldsymbol{x}} \tag{3.2.9}$$

$$\boldsymbol{\tau}_i = \boldsymbol{s}_i'^T \boldsymbol{f}_i \tag{3.2.10}$$

beginning at the end-effector ($i = n$) with starting value $\boldsymbol{f}_n = \boldsymbol{f}_n^*$

4. Dynamic Modeling of Kinematic Loops

This chapter first explains the differences between the dynamic modeling for robots with and without kinematic loops. Then, two different approaches to the dynamic modeling of kinematic loop systems are described in detail. These approaches are based on the general dynamic modeling, which is explained in chapter 3.

4.1. Difficulties of the Dynamic Modeling

Closed kinematic loops occur in many robotic systems as described in chapter 2. To control and simulate those systems, the dynamic modeling of closed kinematic loops is needed. Unfortunately, the dynamic modeling of kinematic loops is much more difficult than the modeling of kinematic tree systems. Since every kinematic loop introduces constraints on the system, kinematic loops comprise more complicated dynamics than kinematic trees. For example there is no correspondence between the degrees of freedom of the system and the joint variables as it is for kinematic trees according to [9]. Furthermore the degrees of motion freedom of mechanisms with kinematic loops can vary, while they stay the same for kinematic tree structures.

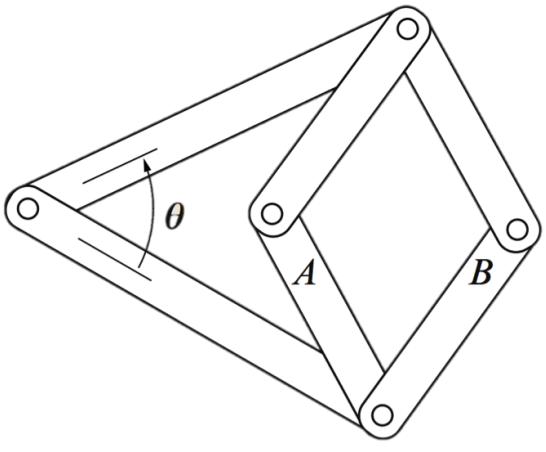


Figure 4.1.: Closed Loop with varying Degrees of Freedom, [8]

Figure 4.1 shows a closed loop system, with varying degrees of freedom. If $\boldsymbol{\theta} \neq 0$, then the system has only one degree of freedom, since the movement of $\boldsymbol{A}$ and $\boldsymbol{B}$ is

depending on the movement of $\boldsymbol{\theta}$. If $\boldsymbol{\theta} = 0$ then $\boldsymbol{A}$ and $\boldsymbol{B}$ can move independently and the system has two degrees of freedom.

Other challenges of closed loops are the kinematic redundancy [28] and the actuation redundancy [29] that has to be dealt with. Furthermore, in kinematic tree structures it is possible to compute all forces whereas not all forces can be computed in closed loops according to [8].

This chapter presents now two different approaches to the computation of forward and inverse dynamics for closed-loop systems.

The first approach was introduced by Nakamura and Yamane [12] as a solution to the dynamics computation of structure-varying mechanisms without switching among algorithms. The method makes use of the generalized coordinates of a kinematic loop. This means that the minimal number of coordinates is used for the computation and results therefore in a highly efficient algorithm. The method aims at the dynamics computation of human figures, nevertheless it is a general approach for closed-loop systems and is therefore described here in detail.

The second approach is a motion simulation method for closed-loop mechanisms described by Featherstone [9]. This forward dynamics computation method is using unknown reaction forces to replace the loop-closing joints. The motion equations for the resulting open-loop system are then computed. Eventually, the acceleration constraints imposed by the closed-loops are added to calculate the dynamics of the whole closed-loop system. After the description of the first approach, this method is described in detail as well.

Similar for both approaches is that the closed loops are first virtually cut for computing the dynamics of the resulting tree structure. Then the constraints equations imposed by the closed loops are added. The differences are that the first approach makes use of generalized coordinates and the second approach is only aiming at the forward dynamics computation.

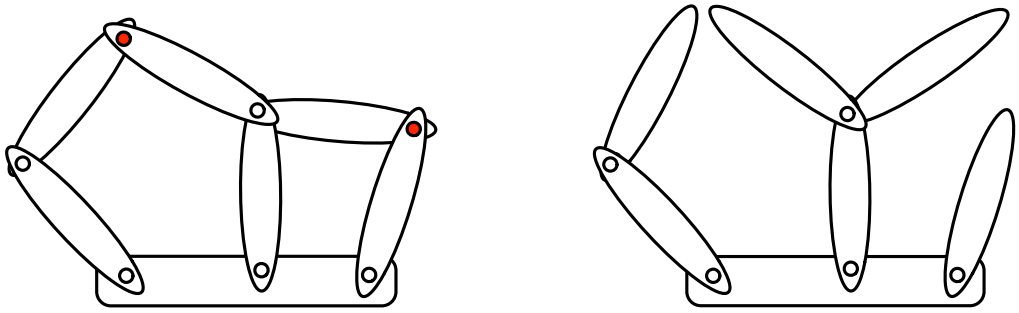
4.2. Dynamic Modeling with Generalized Coordinates

This section describes the dynamic computation of kinematic loops according to Nakamura [12].

4.2.1. Generalized Coordinates of Kinematic Loops

In a closed-loop system as shown in Figure 4.2 (a) the total number of joints is N_J and the joint angles are represented by $\boldsymbol{\theta}_J \in \mathbf{R}^{N_J}$. The number of the actuated joints in the closed loop is N_A, accordingly the actuated joint angles are $\boldsymbol{\theta}_A \in \mathbf{R}^{N_A}$ and the actuator torques are $\boldsymbol{\tau}_A \in \mathbf{R}^{N_A}$.

Now the closed-loop system is virtually cut at joints in such a way that there are no kinematic loops in the system anymore as shown in Figure 4.2 (b). The resulting



(a) Kinematic Loop System (b) Kinematic Tree System

Figure 4.2.: Kinematic Loop and Kinematic Tree System

open-loop system (open kinematic tree-structure) has the number of joints N_O. The joint angles of the tree-structure are $\boldsymbol{\theta}_O \in \mathbf{R}^{N_O}$. All joints are now assumed to be actuated and therefore the joint torques are $\boldsymbol{\tau}_O \in \mathbf{R}^{N_O}$.

The kinematic tree-structure is assumed to perform the same movement as the original closed loop system, however without interaction of the cut joints. With the recursive inverse dynamics algorithm for kinematic trees as described in section 3.2.3, all joint torques $\boldsymbol{\tau}_O$ required for this motion are calculated.

The closed-loop system has N_F (see 4.2.13) degrees of freedom. The generalized coordinates $\boldsymbol{\theta}_G \in \mathbf{R}^{N_F}$ are the independent variables that represent the mobility of the kinematic loop. They describe the motion of the whole closed-loop system with N_F joints. These joints are selected from $\boldsymbol{\theta}_J$ as shown later. Accordingly, the generalized forces $\boldsymbol{\tau}_G \in \mathbf{R}^{N_F}$ are the torques and forces that act on the generalized coordinates.

The movement of the whole closed-loop system is determined by the generalized coordinates $\boldsymbol{\theta}_G$, therefore $\boldsymbol{\theta}_O$ and $\boldsymbol{\theta}_A$ are dependent on $\boldsymbol{\theta}_G$:

$$\boldsymbol{\theta}_O = \boldsymbol{\theta}_O(\boldsymbol{\theta}_G) \tag{4.2.1}$$

$$\boldsymbol{\theta}_A = \boldsymbol{\theta}_A(\boldsymbol{\theta}_G) \tag{4.2.2}$$

With the d' Alembert's principle and the principle of virtual work, the following equation is derived from (4.2.1) as described in [29]:

$$\boldsymbol{\tau}_G^T \delta\boldsymbol{\theta}_G = \boldsymbol{\tau}_O^T \delta\boldsymbol{\theta}_O = \boldsymbol{\tau}_O^T \boldsymbol{W} \delta\boldsymbol{\theta}_G \tag{4.2.3}$$

with

$$\boldsymbol{W} \triangleq \frac{\delta\boldsymbol{\theta}_O}{\delta\boldsymbol{\theta}_G} \in \mathbf{R}^{N_O \times N_F}. \tag{4.2.4}$$

The following equation is derived from (4.2.2) analogously:

$$\boldsymbol{\tau}_G^T \delta\boldsymbol{\theta}_G = \boldsymbol{\tau}_A^T \delta\boldsymbol{\theta}_A = \boldsymbol{\tau}_A^T \boldsymbol{S} \delta\boldsymbol{\theta}_G \tag{4.2.5}$$

with

$$\boldsymbol{S} \triangleq \frac{\delta\boldsymbol{\theta}_A}{\delta\boldsymbol{\theta}_G} \in \mathbf{R}^{N_A \times N_F}. \tag{4.2.6}$$

Eliminating $\delta\boldsymbol{\theta}_G$ in (4.2.3) and in (4.2.5) leads to the following equations:

$$\boldsymbol{\tau}_G = \boldsymbol{W}^T \boldsymbol{\tau}_O \tag{4.2.7}$$

$$\boldsymbol{\tau}_G = \boldsymbol{S}^T \boldsymbol{\tau}_A. \tag{4.2.8}$$

With the equations (4.2.7) and (4.2.8), the actuator torques $\boldsymbol{\tau}_A$ for the kinematic loop can be calculated from $\boldsymbol{W}$, $\boldsymbol{S}$ and the actuator torques of the virtually cut kinematic loop $\boldsymbol{\tau}_O$.

4.2.2. Computation of W and S

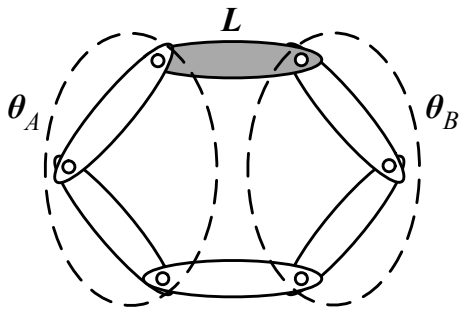


Figure 4.3.: Closed Loop, [12]

In the closed loop in Figure 4.3, the linear and angular velocities of link L are dependent on the linear and angular velocities $\dot{\boldsymbol{\theta}}_A$ and $\dot{\boldsymbol{\theta}}_B$. $\boldsymbol{J}_A$ and $\boldsymbol{J}_B$ are the Jacobian matrices of the position and the orientation of Link L with respect to $\boldsymbol{\theta}_A$ and $\boldsymbol{\theta}_B$. The velocities of Link L are computed by multiplying $\dot{\boldsymbol{\theta}}_A$ or $\dot{\boldsymbol{\theta}}_B$ with the corresponding Jacobian matrix $\boldsymbol{J}_A$ or $\boldsymbol{J}_B$. Due to the closed loop, the resulting velocities of Link L computed from $\dot{\boldsymbol{\theta}}_A$ and $\dot{\boldsymbol{\theta}}_B$ have to be equal:

$$\begin{pmatrix} \boldsymbol{J}_A & -\boldsymbol{J}_B \end{pmatrix} \begin{pmatrix} \dot{\boldsymbol{\theta}}_A \\ \dot{\boldsymbol{\theta}}_B \end{pmatrix} = \mathbf{0}. \tag{4.2.9}$$

This is the velocity constraint enforced by the kinematic loop. In general, the velocity constraint imposed by the ith kinematic loop is

$$\boldsymbol{J}_{Li} \dot{\boldsymbol{\theta}}_J = \mathbf{0} \tag{4.2.10}$$

with $\boldsymbol{J}_{Li} \in \mathbf{R}^{6\times N_J}$. $\boldsymbol{J}_{Li}$ is the velocity constraint matrix of Link L. It is made up of the columns of the Jacobian matrices of link L with respect to the joint angles. The Jacobian matrices are calculated the same way as those for open-loop systems as described in [30].

N_L is the number of independent kinematic loops in the mechanism and therefore N_L constraint matrices exist. $\boldsymbol{J}_C \in \mathbf{R}^{6N_L\times N_J}$ is formed of those constraint matrices as follows:

$$\boldsymbol{J}_C \triangleq \begin{pmatrix} \boldsymbol{J}_{L1} \\ \boldsymbol{J}_{L2} \\ \vdots \\ \boldsymbol{J}_{LN_L} \end{pmatrix}, \tag{4.2.11}$$

so all kinematic constraints are comprised in $\boldsymbol{J}_C$. If not all rows of $\boldsymbol{J}_C$ are independent, then $\boldsymbol{J}_C$ is not full rank and the constraints are dependent. All linearly independent rows of $\boldsymbol{J}_C$ are then selected to form $\boldsymbol{J}_{Cm} \in \mathbf{R}^{m\times N_J}$ with $m = \text{rank}(\boldsymbol{J}_C)$. The m independent constraints introduced by the N_L kinematic loops are represented by

$$\boldsymbol{J}_{Cm}\dot{\boldsymbol{\theta}}_J = \mathbf{0} \tag{4.2.12}$$

which is derived from (4.2.10). From the N_J joints and m independent constraints, the degrees of freedom of the closed kinematic loop structure N_F are calculated:

$$N_F = N_J - m. \tag{4.2.13}$$

$\boldsymbol{J}_{Cm}$ is now divided in $\boldsymbol{J}_S$ and $\boldsymbol{J}_G$ in such a way, that $\boldsymbol{J}_S \in \mathbf{R}^{m\times m}$ contains m linearly independent columns from $\boldsymbol{J}_{Cm}$, while $\boldsymbol{J}_G \in \mathbf{R}^{m\times N_F}$ is built from the remaining ones. Accordingly, $\boldsymbol{\theta}_J \in \mathbf{R}^{N_J}$ is divided in $\boldsymbol{\theta}_S \in \mathbf{R}^m$ and the generalized coordinates $\boldsymbol{\theta}_G \in \mathbf{R}^{N_F}$. For example if the first column of $\boldsymbol{J}_{Cm}$ is independent, it is inserted in $\boldsymbol{J}_S$. Since the first column of $\boldsymbol{J}_{Cm}$ corresponds to the first line of $\boldsymbol{\theta}_J$, the first line of $\boldsymbol{\theta}_J$ is inserted in $\boldsymbol{\theta}_S$. The division of $\boldsymbol{J}_{Cm}$ and $\boldsymbol{\theta}_J$ results in:

$$\boldsymbol{J}_{Cm}\dot{\boldsymbol{\theta}}_J = \begin{pmatrix} \boldsymbol{J}_S & \boldsymbol{J}_G \end{pmatrix} \begin{pmatrix} \dot{\boldsymbol{\theta}}_S \\ \dot{\boldsymbol{\theta}}_G \end{pmatrix} = \mathbf{0}. \tag{4.2.14}$$

Rearranging the terms leads to

$$\boldsymbol{J}_S\dot{\boldsymbol{\theta}}_S = -\boldsymbol{J}_G\dot{\boldsymbol{\theta}}_G. \tag{4.2.15}$$

$\boldsymbol{J}_S$ is always invertible, therefore $\dot{\boldsymbol{\theta}}_S$ can be determined by

$$\dot{\boldsymbol{\theta}}_S = -\boldsymbol{J}_S^{-1}\boldsymbol{J}_G\dot{\boldsymbol{\theta}}_G = \boldsymbol{H}\dot{\boldsymbol{\theta}}_G \tag{4.2.16}$$

with

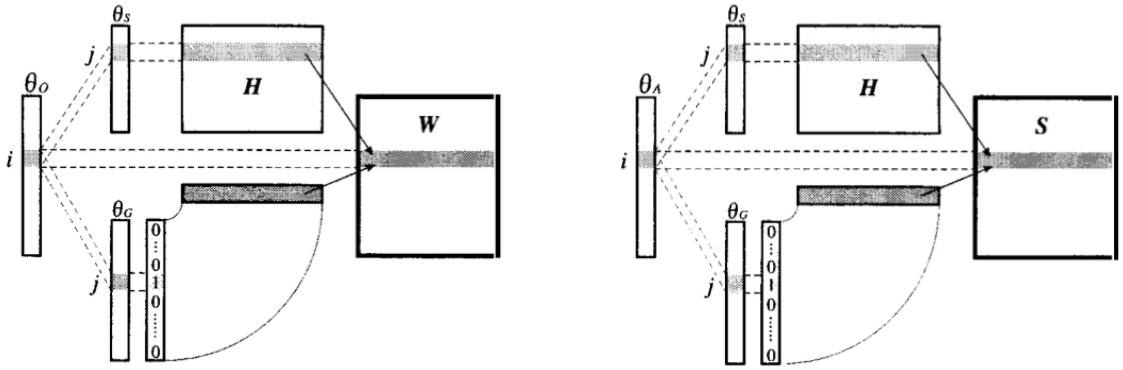
$$\boldsymbol{H} \triangleq \frac{\delta\boldsymbol{\theta}_S}{\delta\boldsymbol{\theta}_G} = -\boldsymbol{J}_S^{-1}\boldsymbol{J}_G. \tag{4.2.17}$$

From $\boldsymbol{H}$ the Jacobian matrices $\boldsymbol{W}$ and $\boldsymbol{S}$ are built according to the following description:

$\boldsymbol{W}$: If the ith joint of $\boldsymbol{\theta}_O$ is not in $\boldsymbol{\theta}_G$ but corresponds with the jth joint of $\boldsymbol{\theta}_S$, then the jth row of $\boldsymbol{H}$ is included as the ith row of $\boldsymbol{W}$. If the ith joint of $\boldsymbol{\theta}_O$ is in $\boldsymbol{\theta}_G$ and corresponds with the jth joint of $\boldsymbol{\theta}_G$, then a vector with the jth element $= 1$ and all other elements $= 0$ is included as ith row of $\boldsymbol{W}$.

$\boldsymbol{S}$ is formed similarly to $\boldsymbol{W}$: If the ith joint of $\boldsymbol{\theta}_A$ is not in $\boldsymbol{\theta}_G$ but corresponds with the jth joint of $\boldsymbol{\theta}_S$, then the jth row of $\boldsymbol{H}$ is included as the ith row of $\boldsymbol{S}$. If the ith joint of $\boldsymbol{\theta}_A$ is in $\boldsymbol{\theta}_G$ and corresponds with the jth joint of $\boldsymbol{\theta}_G$, then a vector with the jth element $= 1$ and all other elements $= 0$ is included as ith row of $\boldsymbol{S}$.

The forming of $\boldsymbol{W}$ and $\boldsymbol{S}$ is visualized in Figure 4.4.



(a) Forming $\boldsymbol{W}$ (b) Forming $\boldsymbol{S}$

Figure 4.4.: Forming of $\boldsymbol{W}$ and $\boldsymbol{S}$, [12]

4.2.3. Relationship of Accelerations

The acceleration of the **dependent** joints $\ddot{\boldsymbol{\theta}}_S$ is needed for the forward dynamics computation. It can be calculated from the generalized acceleration $\ddot{\boldsymbol{\theta}}_G$ by differentiating $\dot{\boldsymbol{\theta}}_S = \boldsymbol{H}\dot{\boldsymbol{\theta}}_G$ (4.2.16):

$$\ddot{\boldsymbol{\theta}}_S = \boldsymbol{H}\ddot{\boldsymbol{\theta}}_G + \dot{\boldsymbol{H}}\dot{\boldsymbol{\theta}}_G \tag{4.2.18}$$

By differentiating $\boldsymbol{H} = -\boldsymbol{J}_S^{-1}\boldsymbol{J}_G$ (4.2.17), the second part of this equation is calculated:

$$\dot{\boldsymbol{H}}\dot{\boldsymbol{\theta}}_G = -\left(\frac{d}{dt}(\boldsymbol{J}_S^{-1})\boldsymbol{J}_G + \boldsymbol{J}_S^{-1}\dot{\boldsymbol{J}}_G\right)\dot{\boldsymbol{\theta}}_G. \tag{4.2.19}$$

Since $\boldsymbol{J}_S^{-1}\boldsymbol{J}_S = \boldsymbol{I}$,

$$\frac{d}{dt}(\boldsymbol{J}_S^{-1})\boldsymbol{J}_S + \boldsymbol{J}_S^{-1}\dot{\boldsymbol{J}}_S = \boldsymbol{0}. \tag{4.2.20}$$

Expanding (4.2.19) leads to:

$$\dot{\boldsymbol{H}}\dot{\boldsymbol{\theta}}_G = -\frac{d}{dt}(\boldsymbol{J}_S^{-1})\boldsymbol{J}_G\dot{\boldsymbol{\theta}}_G - \boldsymbol{J}_S^{-1}\dot{\boldsymbol{J}}_G\dot{\boldsymbol{\theta}}_G.$$

Inserting (4.2.15) results in:

$$\dot{\boldsymbol{H}}\dot{\boldsymbol{\theta}}_G = \frac{d}{dt}(\boldsymbol{J}_S^{-1})\boldsymbol{J}_S\dot{\boldsymbol{\theta}}_S - \boldsymbol{J}_S^{-1}\dot{\boldsymbol{J}}_G\dot{\boldsymbol{\theta}}_G.$$

Rearranging (4.2.20) to $\frac{d}{dt}(\boldsymbol{J}_S^{-1})\boldsymbol{J}_S = -\boldsymbol{J}_S^{-1}\dot{\boldsymbol{J}}_S$ and inserting leaves:

$$\dot{\boldsymbol{H}}\dot{\boldsymbol{\theta}}_G = -\boldsymbol{J}_S^{-1}\dot{\boldsymbol{J}}_S\dot{\boldsymbol{\theta}}_S - \boldsymbol{J}_S^{-1}\dot{\boldsymbol{J}}_G\dot{\boldsymbol{\theta}}_G.$$

With use of (4.2.14), equation (4.2.19) becomes:

$$\dot{\boldsymbol{H}}\dot{\boldsymbol{\theta}}_G = -\boldsymbol{J}_S^{-1}(\dot{\boldsymbol{J}}_S\dot{\boldsymbol{\theta}}_S + \dot{\boldsymbol{J}}_G\dot{\boldsymbol{\theta}}_G) = -\boldsymbol{J}_S^{-1}\dot{\boldsymbol{J}}_{Cm}\dot{\boldsymbol{\theta}}_J. \tag{4.2.21}$$

$\dot{\boldsymbol{J}}_C\dot{\boldsymbol{\theta}}_J$ can be calculated by the same algorithm as for open-loop systems [31]. By extracting linearly independent rows from $\dot{\boldsymbol{J}}_C\dot{\boldsymbol{\theta}}_J$ similar to the extraction of $\boldsymbol{J}_{Cm}$ as described before, $\dot{\boldsymbol{J}}_{Cm}\dot{\boldsymbol{\theta}}_J$ is built.

4.2.4. Inverse Dynamics

The computation of the required actuator torques $\boldsymbol{\tau}_A$ for a given trajectory of the mechanism ($\boldsymbol{\theta}_J, \dot{\boldsymbol{\theta}}_J$ and $\ddot{\boldsymbol{\theta}}_J$) is called inverse dynamics. With the equations described in the preceding sections, the inverse dynamics of closed-loop systems is calculated as follows:

1. The Jacobian matrices $\boldsymbol{W}$ and $\boldsymbol{S}$ are computed according to 4.2.2.
2. The closed loops are virtually cut and $\boldsymbol{\tau}_O$ for kinematic tree structures is calculated with the recursive Newton-Euler inverse dynamics algorithm as described in section 3.2.3.
3. The generalized force $\boldsymbol{\tau}_G$ is computed with $\boldsymbol{W}$ and $\boldsymbol{\tau}_O$ according to (4.2.7).
4. The actuator torques $\boldsymbol{\tau}_A$ are computed with $\boldsymbol{\tau}_G$ by solving (4.2.8):

$$\boldsymbol{\tau}_A = \boldsymbol{S}^{-T}\boldsymbol{\tau}_G \tag{4.2.22}$$

   If the degrees of freedom N_F equals the number of actuators N_A then the system has no actuation redundancy and $\boldsymbol{S} \in \mathbf{R}^{N_A \times N_F}$ becomes a square matrix. $\boldsymbol{\tau}_A$ is then calculated with the inverse of $\boldsymbol{S}$.
   If $N_F \neq N_A$, $\boldsymbol{S}$ is not a square matrix and therefore $\boldsymbol{\tau}_A$ can not be calculated without optimization methods. An optimization method that can be applied to solve $\boldsymbol{\tau}_A$ is described by [32].

4.2.5. Forward Dynamics

Contrary to inverse dynamics, forward dynamics is the computation of the joint accelerations $\ddot{\boldsymbol{\theta}}_J$ for given actuator torques $\boldsymbol{\tau}_A$. The unit vector approach described by [33] is a forward dynamics algorithm intended for kinematic tree structures. This algorithm is extended by the inverse dynamics algorithm described in 4.2.1 for the application to closed-loops systems.

$$\boldsymbol{\tau}_G = \boldsymbol{A}(\boldsymbol{\theta}_G)\ddot{\boldsymbol{\theta}}_G + \boldsymbol{b}(\boldsymbol{\theta}_G, \dot{\boldsymbol{\theta}}_G) \tag{4.2.23}$$

is the equation of motion of closed-loop systems. $\boldsymbol{A} \in \mathbf{R}^{N_F \times N_F}$ is the symmetric inertia matrix. The gravitational, centrifugal and Coriolis effects are specified by $\boldsymbol{b} \in \mathbf{R}^{N_F}$. The computation of the forward dynamics for kinematic loops including the computation of $\boldsymbol{A}$ and $\boldsymbol{b}$ is done as follows:

1. The given actuator torques $\boldsymbol{\tau}_A$ are transformed to the generalized forces $\boldsymbol{\tau}_G$ according to (4.2.8).
2. The inverse dynamics for the generalized acceleration $\ddot{\boldsymbol{\theta}}_G = 0$ is computed and the resulting generalized force is set to $\boldsymbol{b}$. Now the dependent joint accelerations $\ddot{\boldsymbol{\theta}}_S$ are calculated from (4.2.18) with $\ddot{\boldsymbol{\theta}}_G$ being zero: $\ddot{\boldsymbol{\theta}}_S = \dot{\boldsymbol{H}}\dot{\boldsymbol{\theta}}_G$
3. For each degree of freedom ($i = 1, 2, \ldots, N_F$):
   a) The inverse dynamics for $\ddot{\boldsymbol{\theta}}_G = \boldsymbol{e}_i$ with $\boldsymbol{e}_i \in \mathbf{R}^{N_F}$ is computed. $\boldsymbol{e}_i$ is a unit vector with the i-th element being 1 and all other elements being 0. The dependent joint accelerations $\ddot{\boldsymbol{\theta}}_S$ are computed according to (4.2.18). Set the resulting generalized force to $\boldsymbol{f}_i$.
   b) $\boldsymbol{a}_i = \boldsymbol{f}_i - \boldsymbol{b}$ is calculated.
   c) $\boldsymbol{a}_i$ is set as the i-th column of $\boldsymbol{A}$.
4. With $\boldsymbol{\tau}_G$, $\boldsymbol{A}$ and $\boldsymbol{b}$, the generalized acceleration $\ddot{\boldsymbol{\theta}}_G$ is computed by rearranging (4.2.23) to:

$$\ddot{\boldsymbol{\theta}}_G = \boldsymbol{A}^{-1}(\boldsymbol{\tau}_G - \boldsymbol{b}) \tag{4.2.24}$$

5. The dependent joint accelerations $\ddot{\boldsymbol{\theta}}_S$ are computed from (4.2.18) with $\dot{\boldsymbol{H}}\dot{\boldsymbol{\theta}}_G$ already computed in step 2.

With the dependent joint accelerations $\ddot{\boldsymbol{\theta}}_S$ and the generalized joint accelerations $\ddot{\boldsymbol{\theta}}_G$ the accelerations of all joints is known by $\ddot{\boldsymbol{\theta}}_J = \begin{pmatrix} \ddot{\boldsymbol{\theta}}_S \\ \ddot{\boldsymbol{\theta}}_G \end{pmatrix}$

4.3. Dynamic Modeling with Loop Joints

The dynamic modeling of kinematic loops with loop joints according to Featherstone [9] is described in this section.

4.3.1. The Effect of Kinematic Loops on Mechanism Dynamics

According to Featherstone the dynamic model of a closed-loop system is made up of a tree structure system and additional joints, that close the loops. So in a system with N links and L independent kinematic loops there exist $N + L$ joints. The N joints are called tree joints whereas the L joints are called loop joints. As example, in the closed loop in Figure 4.2, there are $N = 6$ tree joints (the one joint connecting 3 links has actually to be modelled as 2 joints) and $L = 2$ loop joints, which are marked red. The system has n degrees of freedom.

A loop joint affects the dynamics of a tree structure system by imposing a constraint on the generalized velocity of the tree structure. For example loop joint k closes the kinematic loop k of an open-loop system by connecting its predecessor link p_k and its successor link s_k. The velocity of s_k is then

$$\boldsymbol{v}_{s_k} = \boldsymbol{v}_{p_k} + \boldsymbol{S}_k \dot{\boldsymbol{q}}_k \tag{4.3.1}$$

with the motion space $\boldsymbol{S}_k$ of joint k and its velocity $\dot{\boldsymbol{q}}_k$. Another expression of the velocity constraint is

$$\boldsymbol{R}_k^T (\boldsymbol{v}_{s_k} - \boldsymbol{v}_{p_k}) = 0 \tag{4.3.2}$$

with the reaction force space $\boldsymbol{R}_k = \boldsymbol{S}_k^{\perp}$ of joint k. The dimension of $\boldsymbol{R}_k$ equals the number of velocity constraints imposed by the closed loop.
The force that joint k applies on link s_k is

$$\hat{\boldsymbol{f}}_k = \boldsymbol{f}_k^a + \boldsymbol{R}_k \boldsymbol{f}_k \tag{4.3.3}$$

with the actuation force $\boldsymbol{f}_k^a$ and the unknown reaction force coefficients $\boldsymbol{f}_k$. Similarly the force that joint k applies on link p_k is $-\hat{\boldsymbol{f}}_k$. Each loop joint introduces r motion constraints and equals the number of unknown reaction forces. The system has $n - r$ degrees of motion freedom if the constraints are independent.

4.3.2. The Equations of Motion for Robots with Kinematic Loops

$$\boldsymbol{H}\ddot{\boldsymbol{q}} = \boldsymbol{Q} - \boldsymbol{C} \tag{4.3.4}$$

is the equation of motion of a general tree structure mechanism with the joint-space intertia matrix $\boldsymbol{H}$, the vector of generalized forces for velocity-product effects like gravity $\boldsymbol{C}$, the vector of generalized forces $\boldsymbol{Q}$ and the accelerations $\ddot{\boldsymbol{q}}$. The motion equations for a system with kinematic loops result from the following method:

1. An acceleration constraint of the kinematic tree is obtained for each loop and expressed in terms of the generalized acceleration.
2. The loop-closure forces are expressed in terms of the generalized forces.

3. The loop-closure forces are added to the motion equation of the kinematic tree structure. Combining them with the acceleration constraints leads to the motion of the closed loop system.

1. Acceleration constraints

By differentiating the velocity constraint equation (4.3.2), the constraint on the acceleration results:

$$\boldsymbol{R}_k^T(\boldsymbol{a}_s - \boldsymbol{a}_p) + \dot{\boldsymbol{R}}_k^T(\boldsymbol{v}_s - \boldsymbol{v}_p) = 0 \tag{4.3.5}$$

where $\boldsymbol{a}_s$ and $\boldsymbol{a}_p$ are the accelerations of link p_k and s_k. Now the acceleration constraint has to be expressed in terms of the generalized acceleration of the kinematic tree $\ddot{\boldsymbol{q}}$.

The velocity of link i in a kinematic tree in terms of the kinematic tree structure velocities is:

$$\boldsymbol{v}_i = \sum_{j=1}^{N} e_{ij}\boldsymbol{S}_j\dot{\boldsymbol{q}}_j \tag{4.3.6}$$

with $\boldsymbol{S}_j$ being the motion space of joint j and its velocity $\dot{\boldsymbol{q}}$. The scalar $e_{ij} = 1$ if joint j is between link i and the base, otherwise $e_{ij} = 0$. A matrix expression of the velocity of link i is:

$$\boldsymbol{v}_i = \boldsymbol{J}_i\dot{\boldsymbol{q}} \tag{4.3.7}$$

with

$$\boldsymbol{J}_i = \begin{bmatrix} e_{i1}\boldsymbol{S}_1 & \dots & e_{iN}\boldsymbol{S}_N \end{bmatrix} \tag{4.3.8}$$

and the generalized velocity of the kinematic tree $\dot{\boldsymbol{q}}$. The expression for the acceleration is given by differentiating (4.3.7):

$$\boldsymbol{a}_i = \boldsymbol{J}_i\ddot{\boldsymbol{q}} + \dot{\boldsymbol{J}}_i\dot{\boldsymbol{q}} \tag{4.3.9}$$

The acceleration of link i due to velocity-product effects is $\dot{\boldsymbol{J}}_i\dot{\boldsymbol{q}}$. This is already calculated for each link to compute $\boldsymbol{C}$ [9, chapter 5] and therefore a known quantity. Now the velocity-product acceleration of link i is called $\boldsymbol{a}_i^{vp} = \dot{\boldsymbol{J}}_i\dot{\boldsymbol{q}}$ and therefore the acceleration of link i is:

$$\boldsymbol{a}_i = \boldsymbol{J}_i\ddot{\boldsymbol{q}} + \boldsymbol{a}_i^{vp} \tag{4.3.10}$$

Inserting (4.3.10) in (4.3.5) leads to

$$\boldsymbol{R}_k^T(\boldsymbol{J}_s - \boldsymbol{J}_p)\ddot{\boldsymbol{q}} = -\boldsymbol{R}_k^T(\boldsymbol{a}_s^{vp} - \boldsymbol{a}_p^{vp}) - \dot{\boldsymbol{R}}_k^T(\boldsymbol{v}_s - \boldsymbol{v}_p) \tag{4.3.11}$$

By defining the velocity across joint k: $\boldsymbol{v}_k = \boldsymbol{v}_s - \boldsymbol{v}_p$, the velocity-product acceleration across joint k: $\boldsymbol{a}_k^{vp} = \boldsymbol{a}_s^{vp} - \boldsymbol{a}_p^{vp}$ and the Jacobian of loop k: $\boldsymbol{J}_k = \boldsymbol{J}_s - \boldsymbol{J}_p$ results

$$\boldsymbol{R}_k^T\boldsymbol{J}_k\ddot{\boldsymbol{q}} = -\boldsymbol{R}_k^T\boldsymbol{a}_k^{vp} - \dot{\boldsymbol{R}}_k^T\boldsymbol{v}_k \tag{4.3.12}$$

This equation expresses the acceleration constraints of the kinematic tree structure introduced by loop k. The dimension of $\boldsymbol{R}_k$ is the same as the number of constraints imposed by the closed loops.

2. Loop-closure forces

To express the loop-closure forces in terms of the equivalent generalized forces it is supposed that a generalized force $\boldsymbol{Q}$ is applied to the tree. $\boldsymbol{Q}$ is applying a force $\boldsymbol{f}$ to link i. For balancing, the force $-f$ must be applied to link i from its predecessors. Therefore each joint between the base and link i applies the force $-f$ on the successor. The generalized force for joint j is then

$$\begin{aligned} \boldsymbol{Q}_j &= -\boldsymbol{S}_j^T \boldsymbol{f}. \\ \boldsymbol{Q} &= -\begin{bmatrix} e_{i1}\boldsymbol{S}_1 & \dots & e_{iN}\boldsymbol{S}_N \end{bmatrix}^T \boldsymbol{f} \end{aligned} \tag{4.3.13}$$

is the generalized force of the whole kinematic tree structure. Using the Jacobian definition (4.3.8) results in

$$\boldsymbol{Q} = -\boldsymbol{J}_i^T \boldsymbol{f}. \tag{4.3.14}$$

If the loop closure force is $\boldsymbol{f}$ on link s_k and $-\boldsymbol{f}$ on link p_k the loop closure forces expressed in equivalent generalized forces is:

$$\boldsymbol{Q}_k^L = \boldsymbol{J}_s^T \boldsymbol{f}_k - \boldsymbol{J}_p^T \boldsymbol{f}_k = \boldsymbol{J}_k^T \boldsymbol{f}_k \tag{4.3.15}$$

The dynamics of the system with closed loop k is modeled by adding $\boldsymbol{Q}_k^L$ to the generalized force. Closing all loops implies the addition of $\sum_{k=1}^{L} \boldsymbol{Q}_k^L$ to the generalized force.

3. Equations of motion

The overall equations of motion of the whole system containing kinematic loops is:

$$\boldsymbol{H}\ddot{\boldsymbol{q}} = \boldsymbol{Q} - \boldsymbol{C} + \sum_{k=1}^{L} \boldsymbol{Q}_k^L \tag{4.3.16}$$

Adding the loop acceleration constraints (4.3.5) and using (4.3.15) and (4.3.3) leads to

$$\begin{bmatrix} \boldsymbol{H} & \boldsymbol{J}_1\boldsymbol{R}_1 & \dots & \boldsymbol{J}_L\boldsymbol{R}_L \\ \boldsymbol{R}_1^T\boldsymbol{J}_1 & 0 & & \\ \vdots & & \ddots & \\ \boldsymbol{R}_L^T\boldsymbol{J}_L & & & \end{bmatrix} \begin{bmatrix} \ddot{\boldsymbol{q}} \\ -\boldsymbol{f}_1 \\ \vdots \\ -\boldsymbol{f}_L \end{bmatrix} = \begin{bmatrix} \boldsymbol{Q} - \boldsymbol{C} + \sum_{k=1}^{L} \boldsymbol{Q}_k^L \\ -\boldsymbol{R}_1^T\boldsymbol{a}_1^{vp} - \dot{\boldsymbol{R}}_1^T\boldsymbol{v}_1 \\ \vdots \\ -\boldsymbol{R}_L^T\boldsymbol{a}_L^{vp} - \dot{\boldsymbol{R}}_L^T\boldsymbol{v}_L \end{bmatrix} \tag{4.3.17}$$

where the coefficient matrix (on the left) $\in \mathbf{R}^{n+r\times n+r}$ with n is the degrees of freedom of the kinematic tree and r the number of constraints imposed by the closed loops and the submatrix $\boldsymbol{H} \in \mathbf{R}^{n\times n}$. Solving the equation for $\ddot{\boldsymbol{q}}$ and the unknown loop closure forces $-\boldsymbol{f}_1 \dots -\boldsymbol{f}_L$ gives the forward dynamics. This is only possible if the coefficient matrix is not singular, otherwise one or more loop closure forces can not be determined. By removing the indeterminate forces, the remaining forces can be calculated. For more detailed explanation see the following example:

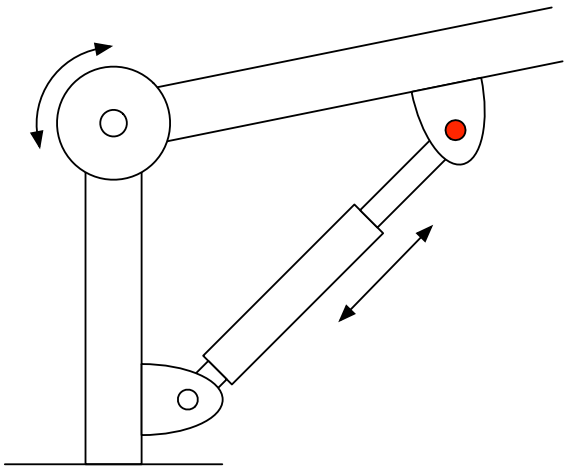


Figure 4.5.: Closed Loop with indeterminate Forces, [9]

The closed loop system in Figure 4.5 has one degree of freedom. If the loop is cut at the red marked joint, the resulting kinematic tree system has three degrees of freedom (one prismatic and two rotational). The red marked loop joint however adds five constraints to the closed loop system. One constraint for each translational movement of the joint in x, y and z direction and one for the rotation around its x and y axis. Only the rotation around its z axis, which points into the picture plane, is possible. Therefore the red marked joint has one degree of freedom and imposes five constraints on the system.

Three of the five constraints are linearly dependent on constraints imposed by the other joints of the system, which allow only planar motion for the system. These three linearly dependent constraints have therefore no effect on the system, since it is already constrained to have planar motion. The two other constraints are independent and therefore affecting the system by closing the loop. Subtracting the linearly independent constraints $r = 2$ from the degrees of freedom of the kinematic tree system $n = 3$ leads to the number of degrees of freedom of the closed loop system, which is 1.

The forces of the three linearly independent constraints can not be calculated and lead the coefficient matrix in 4.3.17 to be singular, therefore they should be removed from the system. To do this, a four degree of freedom joint is used instead of the red marked one degree of freedom joint. This could be modeled by a sphere-in-cylinder joint, that has four degrees of freedom and adds only the two linearly independent constraints. Replacing the rotational joint by the 4 degree of freedom joint, wouldn't change the behaviour of the system and would lead to a nonsingular coefficient matrix, which enables the forward dynamics computation.

5. SpaceDynamics Library

The algorithms described in the preceding chapter 4 have to be implemented in the environment of the SpaceDynamics Library in order to enable the dynamic modeling of robots with closed loops. Since the algorithms makes use of the existing functions, this chapter describes now the structure and the different functions of the SpaceDynamics Library.

Figure 5.1.: Class Diagram of the SpaceDynamics Library – 1

Figure 5.2.: Class Diagram of the SpaceDynamics Library – 2

5.1. Class Diagram

The SpaceDynamics Library is developed at the DLR since 2007 and mainly used for space robot applications. It offers functions that allow the simulation and control of a robot. For example there are functions to calculate the kinematics and dynamics of of a robotic system. All functions of the SpaceDynamics Library are grouped in the six different classes *Model*, *spd*, *matrix*, *vector*,*rot* and *spn*. Figures 5.1 and 5.2 show the class diagram of the library with the associated functions. The library is still under development and some functions are not yet working, or not working for all possible robot configurations. The different classes and some important functions are now explained in detail.

Model

The class *Model* contains all variables to describe a robotic system. In order to use the SpaceDynamics Library functions, the robot has to be represented as an object of this class first. The number of links for example is stored in *Linknum*. The joint configuration of the robot is represented by the joint angles q ($= \boldsymbol{q}$), the joint velocity qd ($= \dot{\boldsymbol{q}}$) and the joint acceleration qdd ($= \ddot{\boldsymbol{q}}$). In general, d in a variable name means derivative. As for example the linear velocity of the robots base is represented by *v0*, whereas its derivative, the linear base acceleration, is stored in *vd0*. A robot can be modeled easily by loading all parameters from a def-file into the SpaceDynamics Library. Table 5.1 gives an overview of the most important variables of the class *Model*.

Table 5.1.: Important Variables of the Class Model

Name	Description
Linknum	number of links in the system
E_Num	number of end effectors
J_type	joint type
JtoC	vector from joint to center of link
CtoJ	vector from center of link to joint
CtoE	vector from center of link to end effector
v0	linear velocity of base v_0
vd0	linear acceleration of base $\dot{v}_0$
w0	angular velocity of base ω_0
wd0	angular acceleration of base $\dot{\omega}_0$
q	joint position $\boldsymbol{q}$
qd	joint velocity $\dot{\boldsymbol{q}}$
qdd	joint acceleration $\ddot{\boldsymbol{q}}$
tau	joint force (torque) $\boldsymbol{\tau}$

spd

The class *spd* offers the main functions of the SpaceDynamics Library, for example *i_dyn_CL_fix()* calculates the inverse dynamics for a fixed base robot model. In general, a *CL* in the function name means, that the function is used for a closed loop robot. In contrast to that, a function without a *CL* in the function name is intended for a kinematic tree system. A *fix* in the function name means, that the function is calculating the solution for a fixed base robot. Functions without *fix* in the function name are intended for free floating base robots.

The differences between functions for fixed base and functions for free floating base are the algorithm and the dimension of the result. For fixed base functions, the robots base has no velocity or acceleration and the movement of the robots links does not affect the pose of the robots base. Therefore the dimension of the output for inverse or forward dynamics is the number of degrees of freedom N_F. For free floating base functions, the movement of the robots links affects the pose of the robots base due to conservation of momentum. Therefore the dimension of

Table 5.2.: Important Functions of the Class spd

Name	Description
model	loads parameters for a robot from a def-file into the model
calc_SP	calculates the spatial notation of the robot
calc_hh	calculates the inertia matrix $\boldsymbol{H}$
calc_Je	calculates the end effector Jacobian
f_kin_e	calculates the pose of the end effector
f_kin_j	calculates the pose of all joints
i_dyn	calculates the inverse dynamics ($\boldsymbol{\tau}$) for an open loop robot with free floating base
i_dyn_fix	calculates the inverse dynamics ($\boldsymbol{\tau}$) for an open loop robot with fixed base
f_dyn	calculates the forward dynamics ($\ddot{\boldsymbol{q}}$) for an open loop robot with free floating base
f_dyn_fix	calculates the forward dynamics ($\ddot{\boldsymbol{q}}$) for an open loop robot with fixed base
i_dyn_CL	calculates the inverse dynamics ($\boldsymbol{\tau}$) for a closed loop robot with free floating base
i_dyn_CL_fix	calculates the inverse dynamics ($\boldsymbol{\tau}$) for a closed loop robot with fixed base
f_dyn_CL_fix	calculates the forward dynamics ($\ddot{\boldsymbol{q}}$) for a closed loop robot with fixed base
H_CL_fix	calculates the inertia matrix $\boldsymbol{H}$ for a closed loop robot with fixed base
dH_CL_fix	calculates matrix $\dot{\boldsymbol{H}}$ for a closed loop robot with fixed base

the output sums up to $N_F + 6$, because the robots base has six degrees of freedom – three translational degrees of freedom in x, y and z direction and three rotational degrees of freedom around those axes.

The functions require the model as input, on which the calculations are performed. The solution of the function is stored in a variable, which is usually received by the function as well. An overview of the most important functions of the class *spd* is given in Table 5.2. These functions are described in detail in section 5.3.

matrix

Functions for basic matrix operations like multiplication of matrices are implemented in the class *matrix*. The functions of this class use functions of the GNU Scientific Library (GSL). These matrix functions are the basis of all implemented algorithms in the SpaceDynamics Library. An overview of the most important functions that are offered by this class is given in Table 5.3.

Table 5.3.: Important Functions of the Class matrix

Name	Description
matrix_get	get a new matrix (memory allocation)
matrix_cpy	copy a matrix
matrix_I	set an identity matrix
matrix_Z	set a zero matrix
matrix_print	print the matrix in matrix form
matrix_mult	multiply two matrices
matrix_add	add two matrices
matrix_sub	subtract a matrix from an other
matrix_trans	transpose matrix
matrix_scale	multiply matrix with scalar
matrix_inv	calculate inverse of matrix
matrix_pinv	calculate pseudoinverse of matrix
matrix_det	return determinant of matrix
matrix_svd	perform a singular value decomposition
matrix_ext_row	extract a row vector from a matrix
matrix_ext_col	extract a column vector from a matrix
matrix_cpy_row	copy a row vector into a matrix
matrix_cpy_col	copy a column vector into a matrix
matrix_ext_sub	extract a submatrix from a matrix
matrix_cpy_sub	copy a submatrix into a matrix

vector

The *vector* class offers similar functions as the *matrix* class. Vectors could also be modeled as one dimensional matrices, however the functions offered by the *vector* class are faster than the *matrix* class functions. Therefore *vector* objects are usually used where it is possible. With the explanation of the functions of the *matrix* class given above, it is easy to understand what a function of the *vector* class is doing. For example function *vector_get* returns a new vector and *vector_cross3* calculates the cross product of two 3×1 matrices. All functions of this class can be found in Figure 5.1.

rot

There are different possibilities to illustrate rotations, for example they can be represented by roll-pitch-yaw angles, direction cosines, rotation matrices and quaternions. The class *rot* offers functions to convert between these different rotation representations. An overview of those functions is given in Table 5.4.

Table 5.4.: Important Functions of the Class rot

Name	Description
rpy2dc	convert roll-pitch-yaw angle to direction cosine
dc2rpy	convert direction cosine to roll-pitch-yaw angle
dc2qtn	convert direction cosine to quaternion
qtn2dc	convert quaternion to direction cosine
rpy2R	convert roll-pitch-yaw angle to rotation matrix
R2rpy	convert rotation matrix to roll-pitch-yaw angle
deg2pi	convert degrees to radians
pi2deg	convert radians to degrees

spn

The class *spn* offers functions for the spatial notation of the robot model. A spatial vector in general is a 6-dimensional vector, that represents the translational and rotational components of a robotic system. With this representation, less equations are needed and it is easier to understand them. The functions calculate for example the coordinate transformation matrix for a rotation around the x,y or z axis with the functions *Xrotx()*, *Xroty()* and *Xrotz()*. These functions are however not used directly by the new algorithms, therefore they are not explained in detail.

5.2. Model Representation

As described before, a model is represented by the variables offered by the *Model* class. A model is defined in an external def-file, which is then loaded into the SpaceDynamics Library. Listing 5.1 shows a minimal model that illustrates the structure of a def-file. The order of the configuration options is fixed and can not be changed. First, parameters of the robot structure are described, then the kinematic parameters are set and finally the dynamics of the system is fixed. After the example def-file, the keywords and parameters are explained.

Listing 5.1: spacedyn/tests/minimal_model_CL.def

```
###Parameters_for_MODEL(__DO_NOT_CHANGE_THIS_ORDER!!___)

###LINK_NUMBER 3

###LINK_CONNECTIVITY
BB[ 0 1 ]

###Joint_Type_[_0=rotational__1=prismatic]
J_type[ 0 0 ]

###EE
EE[ 0 1 ]

###Lflag
CL[ -1 0 ]

###Relative_Coordinate_For_Link:Roll_Pitch_Yaw_Angle_of_Each_Bodies{x-y-z}_[deg]
rpy1[ 0   0   0 ]
rpy2[ 0   0   0 ]

###Vector_Of_Link_Length_JtoC_0_0_is_always_[_0_0_0_]
CtoJ_0_1[ 10   0   0 ]

JtoC_1_1[ 1   0   0 ]
CtoJ_1_2[ 2   0   0 ]

JtoC_2_2[ 3   0   0 ]

###Vector_To_End-Effector
CtoE_1[ 5   0   0 ]

###Relative_Coordinate_For_End-Effector
rpyE_1[ 0 0 0 ]

###Mass_Parameter
mass[ 1 2 3 ]

###Inertia_Matrix
###[_I11_I12_I13]
###[_I21_I22_I23]
###[_I31_I32_I33]

###Link0
I11= 1
I22= 1
I33= 1
I12= 0
I13= 0
I23= 0

###Link1
I11= 1
I22= 1
I33= 1
```

```
55 I12= 0
   I13= 0
57 I23= 0

59 ###Link2
   I11= 1
61 I22= 1
   I33= 1
63 I12= 0
   I13= 0
65 I23= 0

67 ###EOF 777
```

LINK_NUMBER defines the number of links in the robots structure. The value includes the base link. In the example of Listing 5.1 are 3 links.

BB defines the structure of the robot. Elements of this vector define the parent link of link i. The first value defines the parent link of link 1. Listing 5.1 shows a serial kinematic chain. The parent of link 1 is the base link (link 0) and the parent of link 2 is link 1.

J_type describes the type of each joint, starting at joint 1. There are 4 different joint types:

$0 =$ rotational joint around and the z-axis
$1 =$ translational joint along z-axis
$2 =$ rotational joint around custom axis
$3 =$ translational joint along custom axis

For example in Listing 5.1, all joints are rotational.

EE defines which end effector is attached to which link, starting at link 1. 0 means that no end effector is attached to this link. A number e indicates that the end effector e is attached to this link. Thus, only one end effector is possible per link. For example EE[0 0 1 0 2] shows that end effector 1 is connected to link 3, and end effector 2 is connected to link 5. In Listing 5.1 there is only one end effector that is connected to link 2.

CL is the closed loop flag and defines on which link a loop is closed. The value -1 means that this link is not closing a loop. A value l at position i describes that link i closes a loop, by being connected to link l. In Listing 5.1, link 2 is closing the loop by being connected to link 0, the base link.

rpy defines how the link associated coordinate system is rotated around the roll, pitch and yaw angles for each link. The roll, pitch and yaw angles describe the rotation around the x, y and z-axis, respectively. The values are given in degree. For example rpy3 [90 0 0] indicates that the transformation of coordinate system of link 2 to coordinate system of link 3 is a rotation by 90 degrees around the x-axis. To define a custom joint axis, one can use rpy[r p y x y z], were x, y and z define the unit vector of the joint. All joints must have a rpy definition. In the example of Listing 5.1 all link coordinate systems have the same orientation.

CtoJ defines the translational vector from the center of mass of the link to the next joint. JtoJ can be used instead, which defines the translational vector from the previous joint to the following. The values are given in meters. In the example of Listing 5.1, CtoJ_0_1 [10 0 0] means that joint 1 is located 10 meters in x-direction from the center of mass of the base link (link 0).

JtoC defines the translational vector from the joint to the center of mass of the link with values in meters. The CtoJ, JtoJ and JtoC vectors must be in a special order, which is used in Listing 5.1. All joints must have a CtoJ or JtoJ and JtoC vector.

CtoE is the translational vector from the center of mass of the link to the end effector. Similar to CtoJ, all end effectors must have a CtoE definition. In the example of Listing 5.1 the end effector is located 5 meters in x-direction from the center of mass of link 2.

rpyE describes similar to rpy how the end effector associated coordinate system is rotated around the roll, pitch and yaw angles for each end effector with angles in degrees. In Listing 5.1 the end effector is oriented in the same direction as link 2, to which it is connected.

mass defines the mass of each link starting with the base link (link 0). The values are in kg. In the example of Listing 5.1, link 0 has a mass of 1 kg, link 1 has 2 kg and link 2 has 3 kg.

Inertias. The inertia tensor is needed for each link. The parameters are given in kg m^2. The inertia values are listed as described in the minimal model in Listing 5.1. The inertia tensor matrix is given by

$$I = \begin{bmatrix} I_{11} & I_{12} & I_{13} \\ I_{21} & I_{22} & I_{23} \\ I_{31} & I_{32} & I_{33} \end{bmatrix}$$

With all these parameters, a robot can be described and used for calculations with the SpaceDynamics library. The def-file is loaded into a model of the SpaceDynamics Library with the function **model**, which is described in the next section.

5.3. Functions

This section presents now some important functions of the SpaceDynamics Library and describes how they are used.

5.3.1. model

```
void model(string filename, MODEL &m, bool output, bool closed_loop)
```

To load a def-file into the SpaceDynamics Library, the function **model** is used. This function loads the kinematic and dynamic parameters from the def-file into the model and calculates the corresponding spatial notation with **calc_SPN**.

Parameters. The function expects four input parameters:

Table 5.5.: Parameters of the model function

Variable	Type	Description
filename	string	the path to the def-file
m	MODEL	the pointer to the model object to which the parameters are loaded
output	boolean	if true then debug output is printed
closed_loop	boolean	if true then the model contains closed loops

Example. An example how to load the parameters from a def-file into a model is given in the Listing 5.2.

Listing 5.2: Example of Model Loading

```
void load_model(void){
  /* file name */
  string def_file = "LBR3_JtoJ_light.def";
  /* create a new model */
  MODEL m;
  /* load parameters from def-file into model m */
  model(def_file.c_str(), m ,false,false);
}
```

5.3.2. f_kin_e

```
void f_kin_e(MODEL &m, int e_num)
```

The function calculates the forward dynamics of the selected end effector for the current robot configuration. The forward dynamics is the calculation of the pose – that is the position and the orientation of the end effector.

Parameters. The function expects the model and the number of the end effector as input. The orientation and position is then calculated and stored in the model variables m.POS_e and m.ORI_e.

Table 5.6.: Parameters of the f_kin_e function

Variable	Type	Description
m	MODEL	the pointer to the model object
e_num	int	the number of the end effector
m.POS_e[e_num]	Vector3 (3×1)	the position of the end effector e_num
m.ORI_e[e_num]	Matrix3 (3×3)	the rotation matrix of the end effector e_num

Example. An example of this function is given in Listing A.1.

5.3.3. i_dyn_fix

```
void i_dyn_fix(MODEL &m, double *Gravity, double *tau)
```

The function **i_dyn_fix** computes the inverse dynamics for a fixed base multi-body system. This means it calculates the required forces and torques to generate a given motion for a fixed base system. The algorithm implements the recursive Newton-Euler Formulation which is described in section 3.2.3. It has a computational complexity of O(n).

Parameters. The motion of the system is given by the model variables for joint position m.q, velocity m.qd and acceleration m.qdd. The function requires furthermore three parameters: The model m and the gravity are needed as input, tau is the variable where the result is stored.

Table 5.7.: Parameters of the i_dyn_fix function

Variable	Type	Description
m	MODEL	the pointer to the model object
Gravity	double (3×1)	the gravity vector
tau	double ($N_J \times 1$)	the calculated joint forces and torques

Example. An example of this function is given in Listing A.2

5.3.4. f_dyn_fix

```
void f_dyn_fix(MODEL &m, double *Gravity, double *qdd)
```

The function **f_dyn_fix** computes the forward dynamics for a fixed base multi-body system. It calculates the joint accelerations $\ddot{\boldsymbol{q}}$ for a given set of joint forces $\boldsymbol{\tau}$.

Parameters. The input of this function is represented by the model variable for the joint torques m.tau. The function requires furthermore three parameters: The model m and the gravity are needed as input, qdd is the output variable where the result is stored.

Table 5.8.: Parameters of the f_dyn_fix function

Variable	Type	Description
m	MODEL	the pointer to the model object
Gravity	double (3×1)	the gravity vector
qdd	double $(N_J \times 1)$	the calculated joint accelerations

Example. An example of this function is given in Listing A.2

5.3.5. i_dyn

```
void i_dyn(MODEL &m, double *Gravity, double *Force)
```

The function **i_dyn** computes the inverse dynamics for a multi-body system with free floating base, which are usually space robots. In contrast to a fixed base system, the base of a free floating robot can be accelerated. This is done either passively by momentum conservation or actively by applying a torque, for example by using a thruster. The function calculates the required forces of the robots joints and base, that generate a given motion for the system. Here Force[0]- Force[5] represent the forces and torques that are acting on the base, Force[6] - Force[$N_J + 6$] represent the joint forces and torques.

Parameters. The motion of the system is given by the model variables for joint position m.q, velocity m.qd and acceleration m.qdd. The base position, velocity

Table 5.9.: Parameters of the i_dyn_fix function

Variable	Type	Description
m	MODEL	the pointer to the model object
Gravity	double (3×1)	the gravity vector
Force	double $(N_J + 6 \times 1)$	the calculated base and joint forces

and acceleration is set by the 3 dimensional vectors m.POS0, m.v0 and m.vd0, respectively. If the base has a angular velocity or acceleration, this is set by m.w0 and m.wd0. The function requires three parameters: The model m in order to access the model variables and the gravity are needed as input, Force is the variable where the result is stored.

5.3.6. f_dyn

```
void f_dyn(MODEL &m, double *Gravity, double *vd0, double *wd0, double
    *qdd)
```

The function **f_dyn** computes the forward dynamics for a free floating multi-body system. It calculates the joint accelerations and the linear and angular base accelerations for a given set of joint forces, base forces and base torques.

Parameters. The input of this function is represented by the model variables for the joint torques m.tau, the base forces m.F0 and the base torques m.T0. Furthermore, forces (m.Fe) and torques (m.Te) on the end effectors are taken into consideration. The function requires five parameters: The model m and the gravity are needed as input, vd0, wd0 and qdd are the output variables where the result is stored.

Table 5.10.: Parameters of the f_dyn_fix function

Variable	Type	Description
m	MODEL	the pointer to the model object
Gravity	double (3×1)	the gravity vector
vd0	double (3×1)	the calculated linear base acceleration
wd0	double (3×1)	the calculated angular base acceleration
qdd	double $(N_J \times 1)$	the calculated joint accelerations

6. Implementation, Tests and Simulations

This chapter illustrates the implementation and testing of the algorithms to calculate the dynamics of robots with closed loops, which are described in chapter 4. In order to implement the new functions, the existing functions have first to be validated. Then the algorithms for dynamic modeling of robots with closed kinematic loops are implemented and tested. This chapter first shows how the commercial software SIMPACK is used in order to validate the functions of the SpaceDynamics Library. In this scope, the approach to test these functions is described. After that, the implementation, tests and results of the dynamic functions for kinematic trees and for kinematic loops are presented.

6.1. Simpack

SIMPACK is a commercial multi-body simulation software that used for mechanical system design. It is able to simulate the dynamics of different systems for example in aerospace, automotive and rail applications. SIMPACK is used at the DLR for the simulation and demonstration of robotic systems.

Figure 6.1 shows the main window of SIMPACK. One model, that is used for the validation of the SpaceDynamics Library is already displayed on the 3D page. On the right side there is the model tree, which shows the different bodies, joints and other model specific preferences. SIMPACK offers a wide range of options to model a system in a very accurate way.

6.1.1. Test Procedures

Since SIMPACK is a simulation software, it is only capable to compute the forward dynamics – that is the calculation of the joint accelerations $\ddot{\boldsymbol{q}}$ for given joint torques $\boldsymbol{\tau}$. However the inverse dynamics calculation is not possible. Nevertheless both, the inverse and the forward dynamics functions have to be tested for validation.

To test the dynamic functions, the same model must be build in SIMPACK and build as a def-file for the SpaceDynamics Library first. Now, some forces $\boldsymbol{\tau}_{in}$ are applied to the joints in SIMPACK. After the forces simulation, SIMPACK shows the joint accelerations $\ddot{\boldsymbol{q}}_{SP}$ that result due to the applied forces. The same forces $\boldsymbol{\tau}_{in}$ are now applied to the SpaceDynamics Library model. The joint accelerations $\ddot{\boldsymbol{q}}_{SDL}$

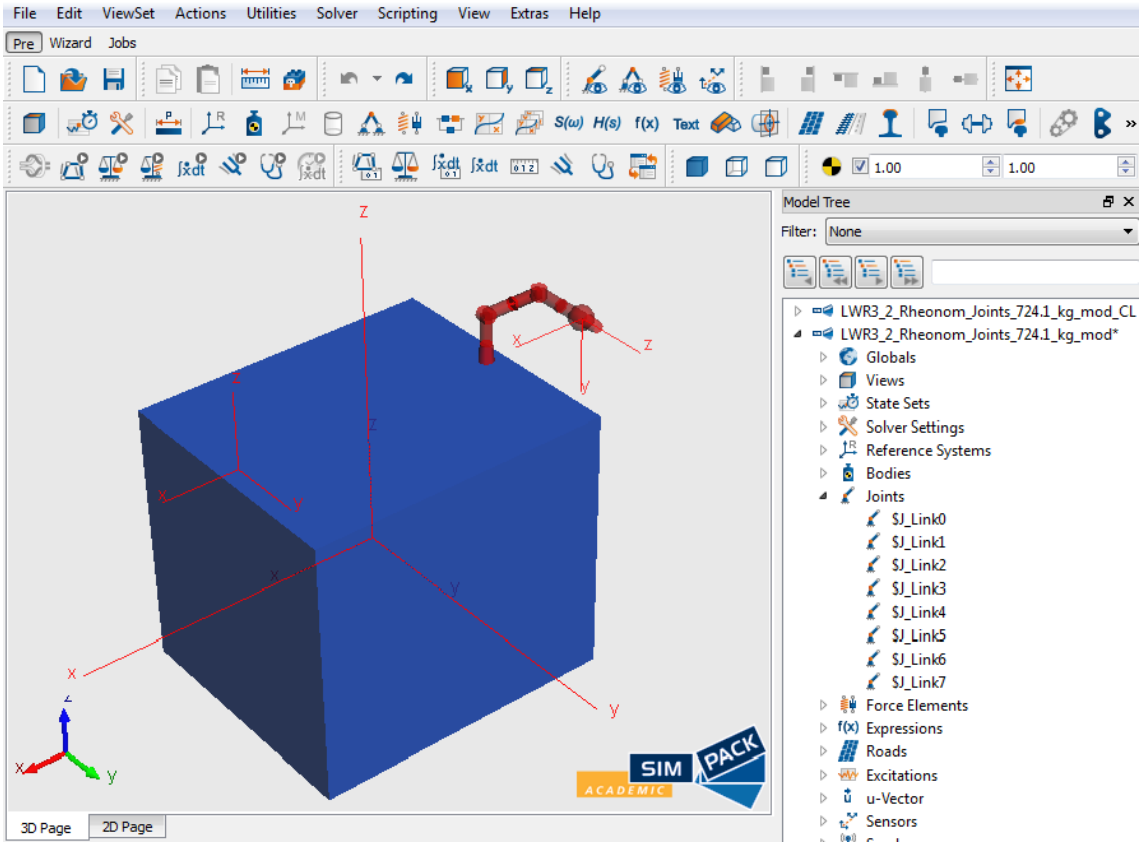
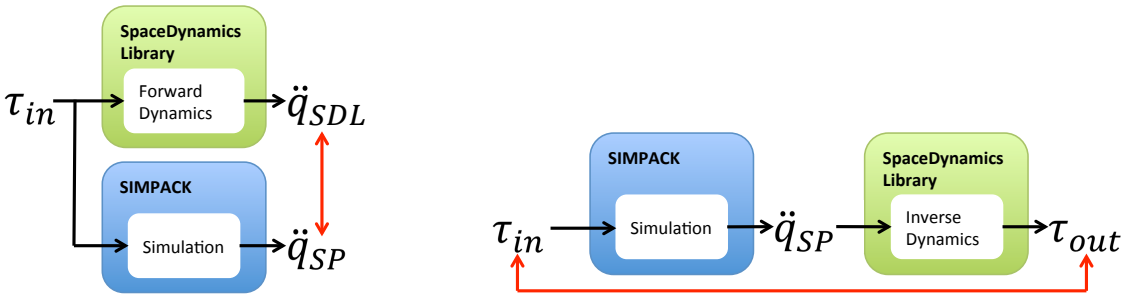


Figure 6.1.: Simpack

are now calculated with the forward dynamics function. If the joint accelerations calculated by SIMPACK $\ddot{\boldsymbol{q}}_{SP}$ correspond with those calculated by the SpaceDynamics Library $\ddot{\boldsymbol{q}}_{SDL}$, the implemented forward dynamics function is valid.

In order to test the inverse dynamics, the joint accelerations $\ddot{\boldsymbol{q}}_{SDL}$ calculated by the forward dynamics are applied to the SpaceDynamics Library model. The joint torques $\boldsymbol{\tau}_{out}$ that are needed to generate those accelerations are calculated with the inverse dynamics. If those calculated joint torques $\boldsymbol{\tau}_{out}$ correspond with the originally applied joint torques $\boldsymbol{\tau}_{in}$, the implemented inverse dynamics function is valid.



(a) Forward Dynamics (b) Inverse Dynamics

Figure 6.2.: Test Procedures

Figure 6.2 illustrates the test procedure. If $\ddot{\boldsymbol{q}}_{SP} = \ddot{\boldsymbol{q}}_{SDL}$, then the forward dynamics function of the SpaceDynamics Library is valid. If $\boldsymbol{\tau}_{in} = \boldsymbol{\tau}_{out}$, then the implemented inverse dynamics function of the SpaceDynamics Library is valid.

6.2. Functions for Kinematic Trees

6.2.1. Test of Forward Kinematics

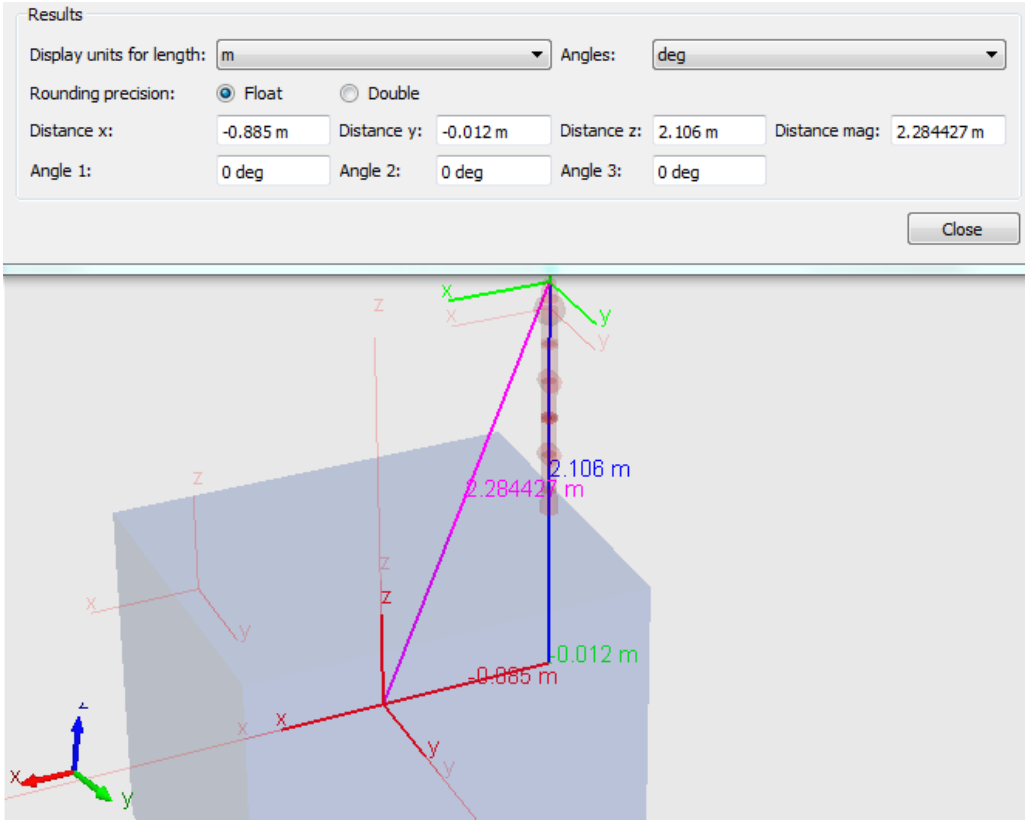
Before implementing the new algorithms, the existing functions of the SpaceDynamics Library had to be tested. A forward kinematics test was implemented in order to test if the def-file model corresponds with the model built in SIMPACK. As described in the introduction, forward kinematics is the calculation of the pose of the end effector for given joint positions. In the appendix, the implemented test function is shown in Listing A.1.

Figures 6.3 and 6.4 show the tests for different configurations. In the Forward Kinematics Test 1, the model is in its initial setup. All joint angles are 0. With the *measure tool*, the pose of the end effector is measured in SIMPACK. The SpaceDynamics Library uses the forward kinematics function **f_kin_e** (section 5.3.3) in order to determine the pose of the end effector. The result of the Forward Kinematic Test 1 are shown in the Figure 6.3 and Table 6.1 and are identical.

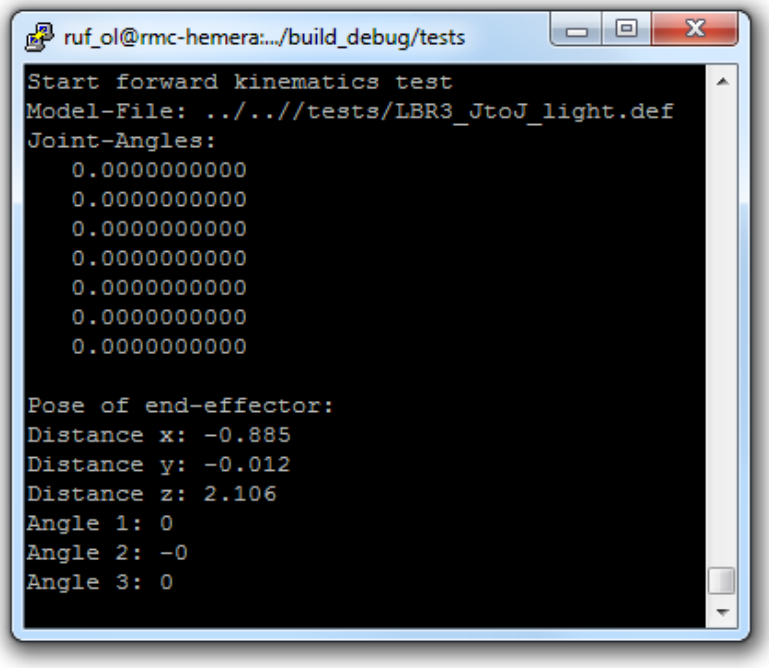
Table 6.1.: Results of Forward Kinematics Test 1

Variable	SIMPACK	SDL
Distance x	-0.885 m	-0.885 m
Distance y	0.012 m	0.012 m
Distance z	2.106 m	2.106 m
Angle 1	0 deg	0 deg
Angle 2	0 deg	0 deg
Angle 3	0 deg	0 deg

To be sure that the model representation in SIMPACK is corresponding with the SpaceDynamics Library, a second forward kinematics test is performed. This time, the joint angles of joint 2,3 and 4 are each set to $90deg$. The results are shown in Figure 6.4 and Table 6.2. As for the first test, the results correspond with each other. Therefore the forward kinematics method is valid. The model representation in both, SIMPACK and the SpaceDynamics Library correspond with each other. For this reason, the model can be used to perform the following tests. The def-file of the model is displayed in Listing A.5.

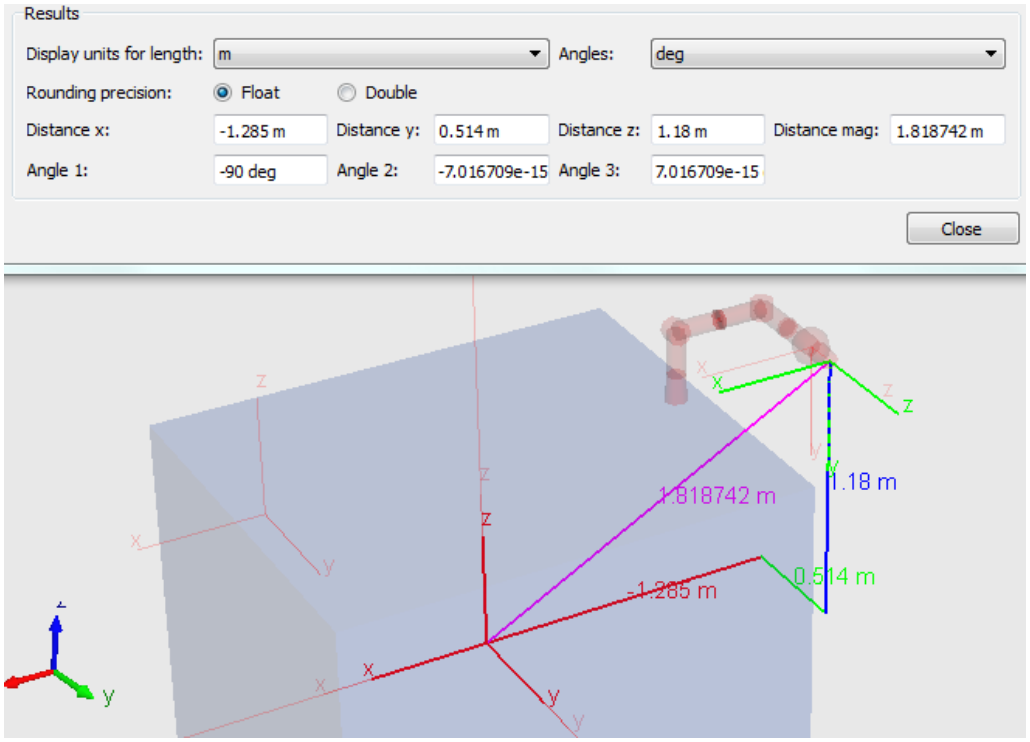


(a) SIMPACK

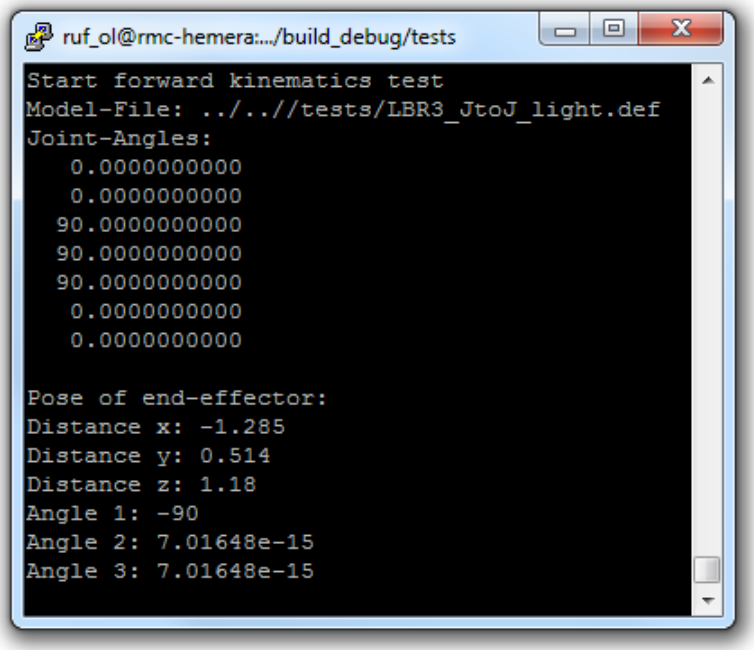


(b) SpaceDynamics Library

Figure 6.3.: Forward Kinematics Test 1



(a) SIMPACK



(b) SpaceDynamics Library

Figure 6.4.: Forward Kinematics Test 2

Table 6.2.: Results of Forward Kinematics Test 2

Variable	SIMPACK	SDL
Distance x	-1.285 m	-1.285 m
Distance y	0.514 m	0.514 m
Distance z	1.18 m	1.18 m
Angle 1	-90 deg	-90 deg
Angle 2	7.016e-15 deg	7.016e-15 deg
Angle 3	7.016e-15 deg	7.016e-15 deg

6.2.2. Test of Dynamics for Fixed Base

After the successful forward kinematics test, the forward and inverse dynamic functions for a kinematic tree system with fixed base were tested. These methods especially the function **i_dyn_fix** (see 5.3.3) is needed for the dynamics calculation of closed loops.

For the test, the same robot configuration as in the Forward Kinematics Test 2 was used. Additionally, an initial velocity was set to joint 2 and different torques were set to joint 2 and 4. The parameters of the robot configuration are shown in Table 6.3 and the implemented test is printed in Listing A.2.

Table 6.3.: Model Configuration of Dynamic Test

Variable	Value	Description
m	LBR3_JtoJ_light.def	Model
m.q[2]	90 deg	joint 2 angle
m.q[3]	90 deg	joint 3 angle
m.q[4]	90 deg	joint 4 angle
m.qd[2]	$-1\frac{\mathrm{m}}{\mathrm{s}}$	joint 2 velocity
m.tau[2]	1 Nm	joint 2 torque
m.tau[4]	-1 Nm	joint 4 torque

By analyzing the results of the SpaceDynamics Library which are displayed in Figure 6.5 (b) and Table 6.5, one can easily identify an error. The input of the forward dynamics function are the joint torques, as shown in Table 6.3. With these torques, the forward dynamics calculates the resulting joint accelerations. These joint accelerations are now taken as the input of the inverse dynamics. Now, the inverse dynamics calculates the joint torques that generate those accelerations. Unfortunately, the input joint torques $\boldsymbol{\tau}_{in}$ do not correspond with the the calculated joint torques $\boldsymbol{\tau}_{out}$. Therefore an error is in the code.

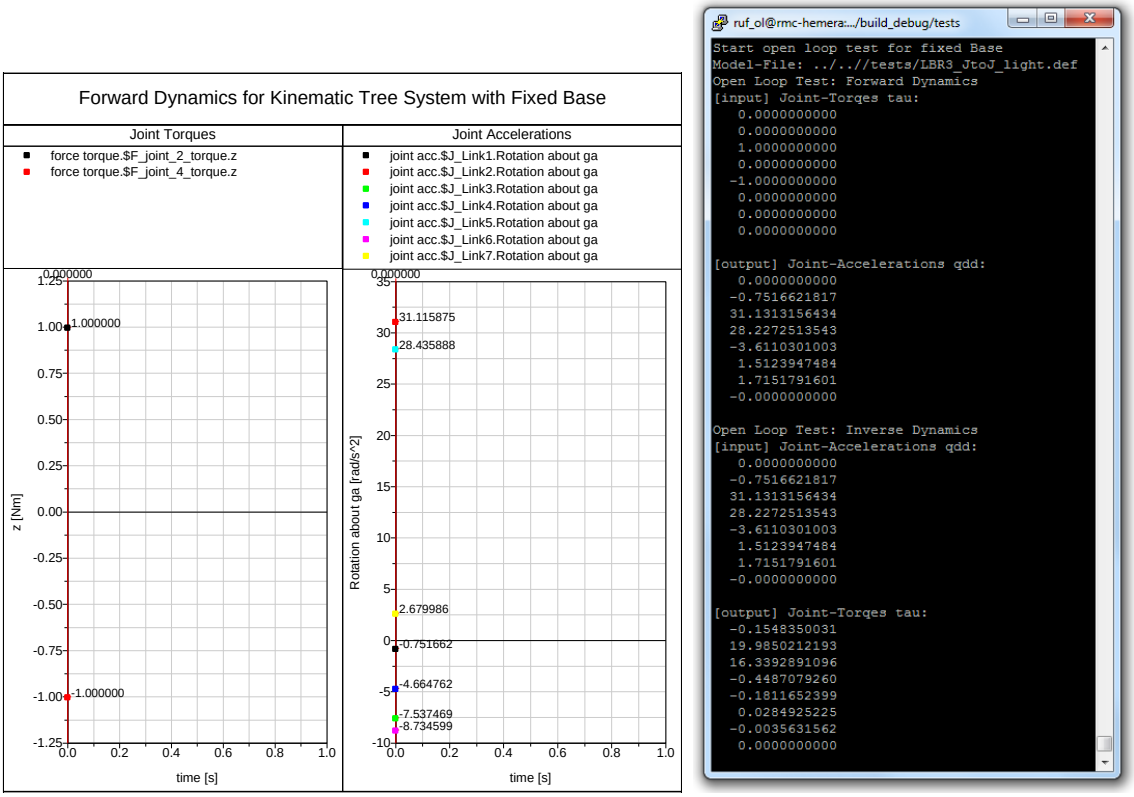
Now, the results of the SIMPACK simulation and the SpaceDynamics library forward dynamic function are compared. They are shown in Figure 6.5 and in Table 6.4 The acceleration of joint 1 and joint 2 corresponds with both functions. The calculated accelerations of the other joints do however not correspond with each

Table 6.4.: Results of the Forward Dynamics Test

Joint	$\ddot{q}_{SP}$ $\left[\frac{\mathrm{m}}{\mathrm{s}^2}\right]$	$\ddot{q}_{SDL}$ $\left[\frac{\mathrm{m}}{\mathrm{s}^2}\right]$
0	0	0
1	-0.751662	-0.7516621817
2	31.115875	31.1313156434
3	-7.537469	28.2272513543
4	-4.664762	-3.6110301003
5	28.435888	1.5123947484
6	-8.734599	1.7151791601
7	2.679986	0

Table 6.5.: Results of the Inverse Dynamics Test

Joint	τ_{in} [Nm]	τ_{out} [Nm]
0	0	-0.1548350031
1	0	19.9850212193
2	1	16.3392891096
3	0	-0.4487079260
4	-1	-0.1811652399
5	0	0.0284925225
6	0	-0.0035631562
7	0	0



(a) SIMPACK (b) SpaceDynamics Library

Figure 6.5.: Dynamics Test for Kinematic Tree with Fixed Base

other. At this point, it is clear that at least one error exists in the forward dynamics function.

6.2.3. Matrix Based Dynamics for Fixed Base

In order to find the error of the implemented forward dynamics algorithm, the new functions **i_dyn_fix_matrix_based** and **f_dyn_fix_matrix_based** were implemented.

```
void i_dyn_fix_matrix_based(MODEL &m, double *Gravity, double *tau)

void f_dyn_fix_matrix_based(MODEL &m, double *Gravity, double *qdd)
```

The functions **i_dyn_fix_matrix_based** and **f_dyn_fix_matrix_based** compute the inverse and forward dynamics similar to the function **i_dyn_fix** (see 5.3.1) and **f_dyn_fix**. Instead of using the recursive Newton-Euler Formulation, these algorithms implement equation 3.1.1 directly and have therefore a computational complexity of $O(n^4)$. The parameters of **i_dyn_fix_matrix_based** are exactly the same as described for the function **i_dyn_fix**. The implemented functions are shown in Listings A.3 and A.4.

With these functions it is possible to test the dynamic model representation, for example if the inertia matrix $\boldsymbol{H}$ is calculated appropriately. The implementation of the test equals the one of the dynamics for a fixed base system (see Listing A.2). Except the functions **i_dyn_fix** and **f_dyn_fix** were replaced by **i_dyn_fix_matrix_based**

Table 6.6.: Results of the Forward Dynamics Test

Joint	$\ddot{q}_{SP}$ $[\frac{m}{s^2}]$	$\ddot{q}_{SDL}$ $[\frac{m}{s^2}]$
1	-0.751662	-0.7516621817
2	31.115875	31.1158748271
3	-7.537469	-7.5374685603
4	-4.664762	-4.6647625308
5	28.435888	28.4358887822
6	-8.734599	-8.7345991083
7	2.679986	2.6799860449

Table 6.7.: Results of the Inverse Dynamics Test

Joint	τ_{in} [Nm]	τ_{out} [Nm]
1	0	-0
2	1	1
3	0	0
4	-1	-1
5	0	0
6	0	0
7	0	0

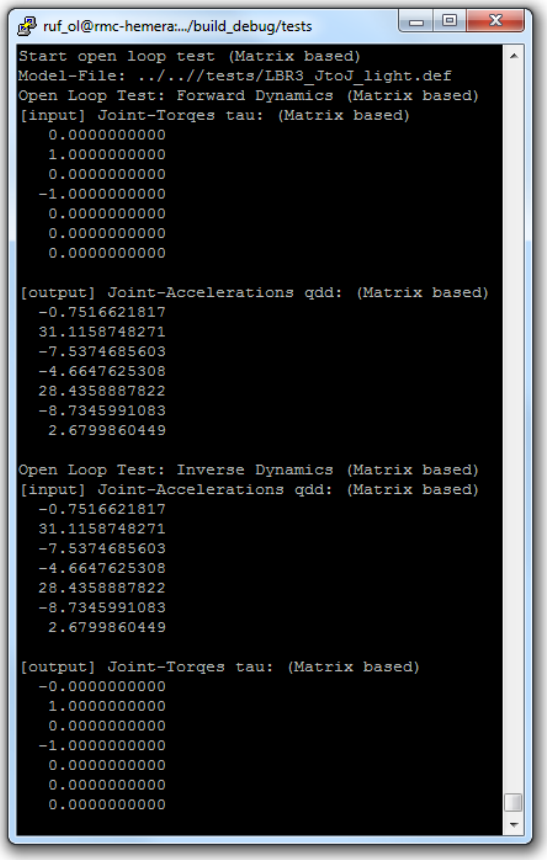


Figure 6.6.: SpaceDynamics Library

and **f_dyn_fix_matrix_based**, respectively.

For the test, the same parameters as described in Table 6.3 are used. The results of the matrix based calculations are shown in Figure 6.6. The results of the SIMPACK simulation (see Figure 6.5) are compared to the results of the forward dynamics matrix based test in Table 6.7. The results of the inverse dynamics is shown in Table 6.7. Since $\ddot{\boldsymbol{q}}_{SP} = \ddot{\boldsymbol{q}}_{SDL}$ and $\boldsymbol{\tau}_{in} = \boldsymbol{\tau}_{out}$, the both implemented functions, **i_dyn_fix_matrix_based** and **f_dyn_fix_matrix_based** are valid.

The results also prove the dynamic model representation of the SpaceDynamics Library to be valid. Therefore, the error of the forward dynamics function **f_dyn_fix** is not because of a wrong calculation of the inertia matrix $\boldsymbol{H}$ for example, but rather because of an error in the code.

6.2.4. Dynamics for Fixed Base – Error Correction

The error in the forward dynamics function for fixed base robots was found after debugging the code step by step. In Listing 6.1 one can detect the error.

Listing 6.1: spacedyn/src/f_dyn_fix.cpp

```
250        qdd[i] = ( u[i] - (*tmp3) + (*tmp4) ) * (d_recip);

252        matrix_scale( 6, 1, m.qdd[i], m.S[i], tmp );
```

In line 250 the variable *qdd* is set which represents the output. In the next step, *qdd* should be multiplied with a scalar value and stored in the variable *tmp*. However, not *qdd* but *m.qdd*, which is not yet set, is multiplied. This leads to a wrong value of *tmp*, which has a big effect on the further calculation. By replacing *m.qdd* with *qdd*, the error is removed. Now the forward dynamics function **f_dyn_fix** obtains the very same results as the function **f_dyn_fix_matrix_based**, which are displayed in Figure 6.6 and Tables 6.6 and 6.7. Since $\ddot{\boldsymbol{q}}_{SP} = \ddot{\boldsymbol{q}}_{SDL}$ and $\boldsymbol{\tau}_{in} = \boldsymbol{\tau}_{out}$, the functions **f_dyn_fix** and **i_dyn_fix** are now valid as well.

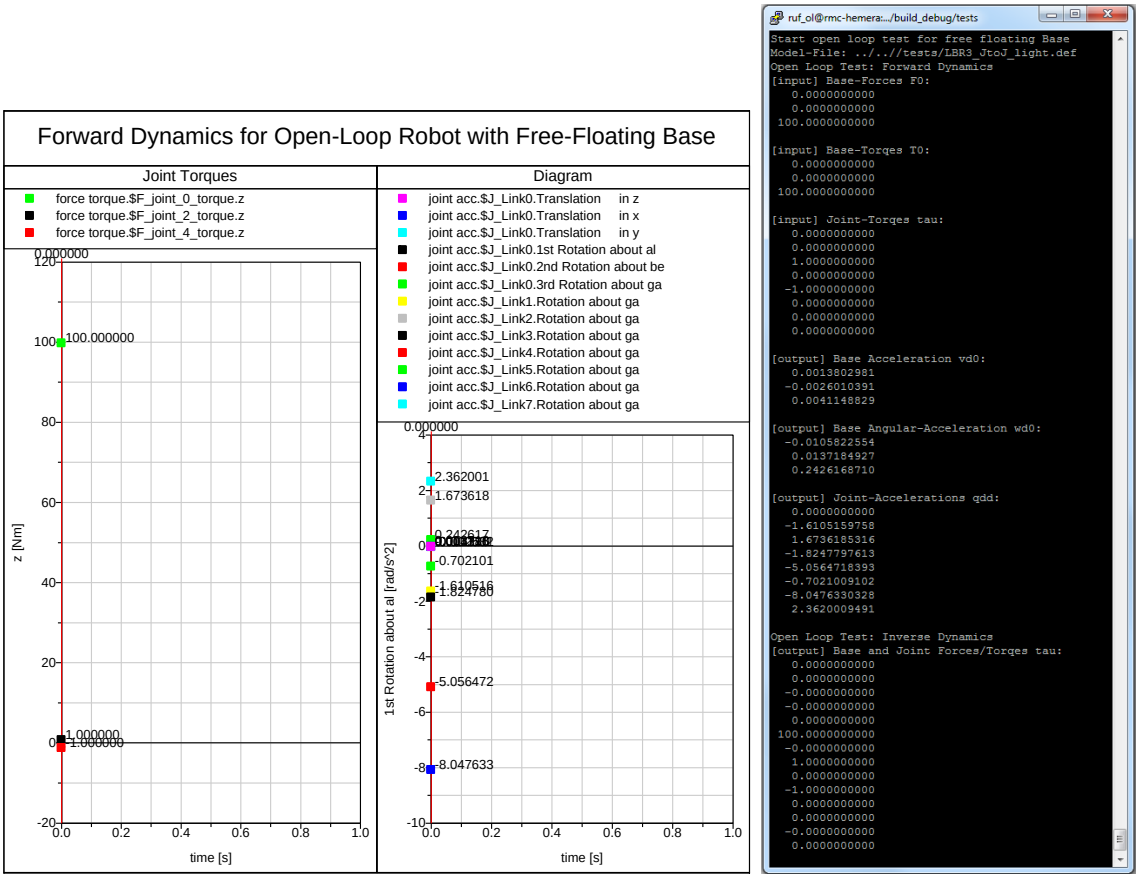
Of course this was not the only error that occurred during the testing and the implementation of the new algorithms. The description of the error, the implementation of other functions as temporary solutions and the error correction shows exemplary how errors have been detected and corrected in general.

6.2.5. Dynamics for Free Floating Base

After the error in the function **f_dyn_fix** was detected and corrected, the next functions to test are the forward and inverse dynamics for free floating base robots – **f_dyn** and **i_dyn** (see sections 5.3.5 and 5.3.6). The tests are performed similar to the tests of the dynamics functions for fixed base robots. This is the same implementation as in Listing A.2, except the functions **f_dyn_fix** and **i_dyn_fix** are replaced by the functions **f_dyn** and **i_dyn**, respectively. The robot configuration for this test is shown in Table 6.8.

Table 6.8.: Model Configuration of Dynamic Test

Variable	Value	Description
m	LBR3_JtoJ_light.def	Model
m.q[2]	90 deg	joint 2 angle
m.q[3]	90 deg	joint 3 angle
m.q[4]	90 deg	joint 4 angle
m.qd[2]	$-1\frac{\mathrm{m}}{\mathrm{s}}$	joint 2 velocity
m.T0[2]	100 Nm	z-axis base torque
m.tau[2]	1 Nm	joint 2 torque
m.tau[4]	-1 Nm	joint 4 torque



(a) SIMPACK (b) SpaceDynamics Library

Figure 6.7.: Dynamics Test for Kinematic Tree with Free-Floating Base

The same error that was in the function **f_dyn_fix**, also occurred in function **f_dyn**. Furthermore the gravity was defined in the wrong direction. After removing these errors, the output of the test is shown in Figure 6.7 and Tables 6.9 and 6.10. The base linear acceleration is represented by vd0 whereas the base angular acceleration is represented by wd0. The force that acts on the base is represented by F0, the torque by T0.

Table 6.9.: Results of the Forward Dynamics Test

Joint	$\ddot{q}_{SP}$ $[\frac{m}{s^2}]$	$\ddot{q}_{SDL}$ $[\frac{m}{s^2}]$
vd0[x]	0.0013803	0.0013802981
vd0[y]	-0.00260104	-0.0026010391
vd0[z]	0.00411488	0.0041148829
wd0[x]	-0.0105823	-0.0105822554
wd0[y]	0.0137185	0.0137184927
wd0[z]	0.242617	0.2426168710
1	-1.61052	-1.6105159758
2	1.67362	1.6736185316
3	-1.82478	-1.8247797613
4	-5.05647	-5.0564718393
5	-0.702101	-0.7021009102
6	-8.04763	-8.0476330328
7	2.362	2.3620009491

Table 6.10.: Results of the Inverse Dynamics Test

Joint	τ_{in} [Nm]	τ_{out} [Nm]
F0[x]	0	0
F0[y]	0	0
F0[z]	0	-0
T0[x]	0	-0
T0[y]	0	0
T0[z]	100	100
1	0	-0
2	1	1
3	0	0
4	-1	-1
5	0	0
6	0	0
7	0	-0

All results obtained by the SpaceDynamics Library correspond with the results obtained by the SIMPACK simulation. Therefore the functions **f_dyn** and **i_dyn** are valid.

6.3. Functions for Kinematic Loops

After successful testing of the dynamic functions for kinematic trees, the functions for kinematic loops were implemented. First, one algorithm of those presented in chapter 4, had to be chosen. The dynamic modeling with generalized coordinates by Nakamura [12] (see section 4.2.1) offers a forward and inverse dynamics method, whereas the dynamic modeling with loop joints by Featherstone [9] (see section 4.3) offers only a method to compute the forward dynamics. Nakamuras approach uses the generalized coordinates of a kinematic loop, which is the minimal number of coordinates that is needed for the computation. The method is therefore computational efficient. Although the method aims at the application to human figures, it is a general approach to the dynamic modeling of kinematic loops. Furthermore the algorithm offers the dynamic modeling for robots with free floating base and varying structure. For these reasons the algorithm of Nakamura was chosen to be implemented in the SpaceDynamics Library. The dynamic modeling method with generalized coordinates is distributed over several functions, which are explained now in detail.

6.3.1. H_CL_fix

```
double *H_CL_fix( MODEL &m, int *index_m, int *index_n , unsigned int *
    Nf, unsigned int **q_S, unsigned int **q_G){
```

The function **H_CL_fix** calculates the Jacobian matrices $\boldsymbol{J}_C$, $\boldsymbol{J}_{Cm}$, $\boldsymbol{J}_S$, $\boldsymbol{J}_G$ and $\boldsymbol{H}$ according to the algorithm presented in section 4.2.1 for a fixed base robotic system. This function is used by the forward dynamics and inverse dynamics computation. Equations 4.2.11 - 4.2.17 are implemented.

Parameters. The function expects six parameters. The only input parameter is the model, the other parameters are output variables. Table 6.11 gives an overview of the parameters for this function. The function returns the pointer to matrix $\boldsymbol{H}$.

Table 6.11.: Parameters of the H_CL_fix function

Variable	Type	Description
m	MODEL	the pointer to the model object
index_m	int	the number of lines of $\boldsymbol{H}$
index_n	int	the number of columns of $\boldsymbol{H}$
Nf	int	the degrees of freedom N_F
q_S	int (index_m $\times$ 1)	the index of the dependent joint angles $\boldsymbol{\theta}_S$
q_G	int (index_n $\times$ 1)	the index of the generalized coordinates $\boldsymbol{\theta}_G$

6.3.2. i_dyn_CL_fix

```
void i_dyn_CL_fix( MODEL &m, double *Gravity, double *Torque_CL, double
    *p_taudist)
```

The function **i_dyn_CL_fix** computes the inverse dynamics for a kinematic loop system with fixed base. It calculates the required actuator torques in order to perform a certain motion. Similar to the function **i_dyn_fix** for kinematic tree systems, the motion is given by the joint positions, velocities and accelerations. The method implements the inverse dynamics of the algorithm as described in section 4.2.4 and uses **i_dyn_fix**.

Parameters. The function expects four parameters. The Model, the gravity vector and the torque distribution vector are the inputs, Torque_CL is output variables containing the result. Table 6.12 gives an overview of the parameters for this function.

Table 6.12.: Parameters of the i_dyn_CL_fix function

Variable	Type	Description
m	MODEL	the pointer to the model object
Gravity	double (3 $\times$ 1)	the gravity vector
p_taudist	double ($N_A - N_F \times 1$)	the torque distribution vector
Torque_CL	double ($N_A \times 1$)	the calculated actuator torques

6.3.3. f_dyn_CL_fix

```
void f_dyn_CL_fix( MODEL &m, double *Gravity, double *qdd)
```

The function **f_dyn_CL_fix** computes the forward dynamics for a kinematic loop system with fixed base. It calculates the joint accelerations for a given set of actuator torques. The method implements the inverse dynamics of the algorithm as described in section 4.2.5 and uses **i_dyn_CL_fix**.

Parameters. The function expects three parameters. The Model and the gravity vector are the inputs, qdd is the output variable, that is containing the result. Table 6.13 gives an overview of the parameters for this function.

Table 6.13.: Parameters of the f_dyn_CL_fix function

Variable	Type	Description
m	MODEL	the pointer to the model object
Gravity	double (3×1)	the gravity vector
qdd	double ($N_J \times 1$)	the calculated joint accelerations

6.3.4. Test of Dynamics for Fixed Base – 1

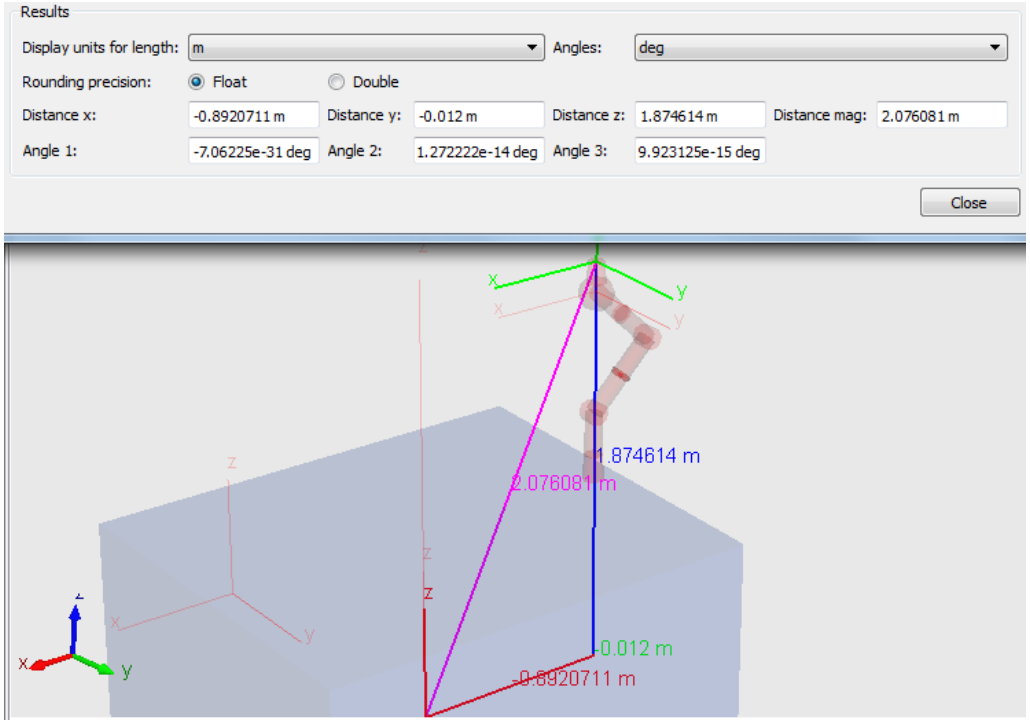
The implemented dynamic functions for closed loop robots are now tested. To create a closed loop, the same system is used as for the open loop tests, but the end effector is set to a fixed position. The motion of the end effector is constrained in 6 dimensions. So no linear movement along and no rotation around the x, y and z-axis is possible. In the def-file, this is done by adding a line with the closed loop flag. The def-file of the used model is displayed in Listing A.6. The loop is closed on Link 7. The robot configuration is shown in Table 6.14.

Table 6.14.: Model Configuration of Dynamic Test

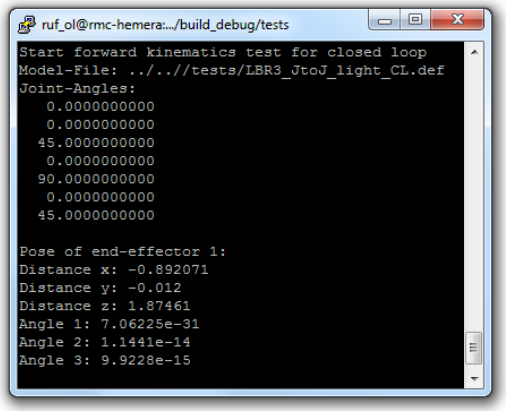
Variable	Value	Description
m	LBR3_JtoJ_light_CL.def	Model
m.q[2]	45 deg	joint 2 angle
m.q[4]	90 deg	joint 4 angle
m.q[6]	45 deg	joint 6 angle
m.tau[1]	1 Nm	joint 1 torque

First, a forward kinematics test is performed, in order to test the model representation in SIMPACK and the SpaceDynamics library. The results of the tests are displayed in Figure 6.8 and Table 6.15.

The results obtained by SIMPACK, correspond with the results from the SpaceDynamics Library function, therefore the model representation is valid and can be used for further testing.



(a) SIMPACK



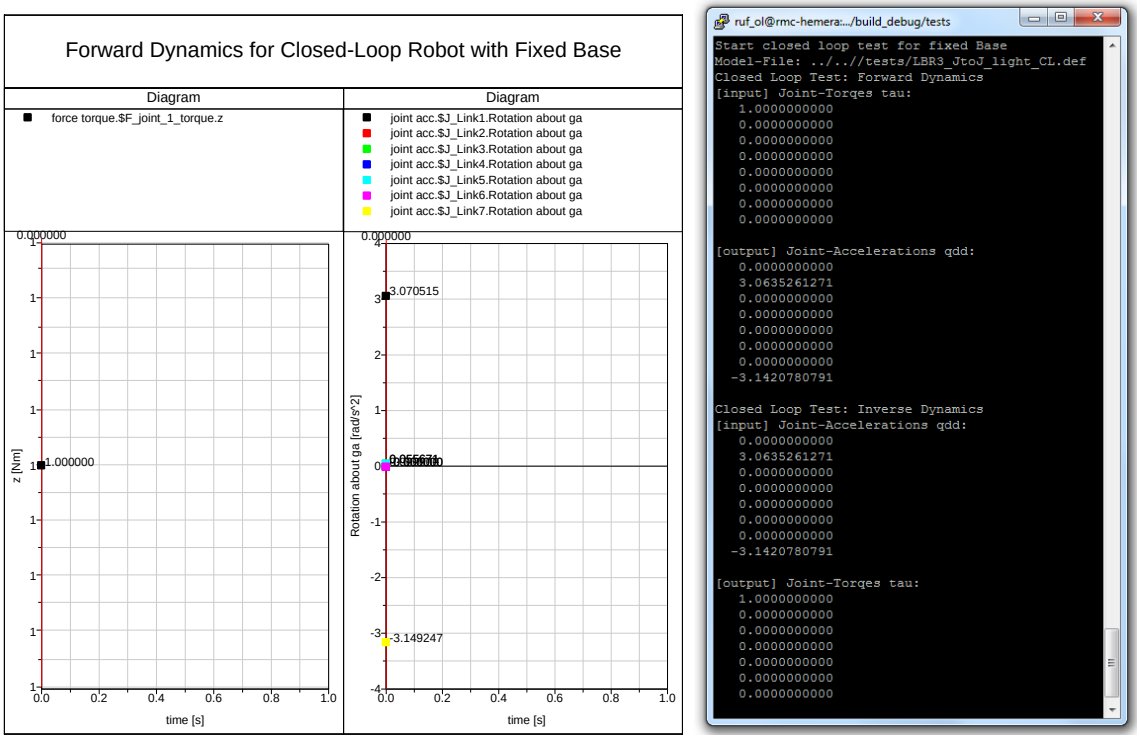
(b) SpaceDynamics Library

Figure 6.8.: Results of Forward Kinematics Test

Table 6.15.: Results of Forward Kinematics Test

Variable	SIMPACK	SDL
Distance x	-0.8920711 m	-0.892071 m
Distance y	-0.012 m	-0.012 m
Distance z	1.874614 m	1.87461 m
Angle 1	7.06225e-31 deg	7.06225e-31 deg
Angle 2	1.272222e-14 deg	1.1441e-14 deg
Angle 3	9.923125e-15 deg	9.9228e-15 deg

Now, the forward and inverse dynamics of the system are tested. The test is implemented similar to the dynamic tests for the kinematic tree system, which is shown by Listing A.2. However, the model configuration is not the same and the functions **f_dyn_fix** and **i_dyn_fix** are replaced by **f_dyn_CL_fix** and **i_dyn_CL_fix**. The results are are shown in Figure 6.9 and in Tables 6.16 and 6.17. The results of the forward dynamics differ by only 0.228%, they are therefore corresponding with each other. The inverse dynamics function calculates the very same result as the SIMPACK simulation. The functions are therefore considered to be valid.



(a) SIMPACK (b) SpaceDynamics Library

Figure 6.9.: Dynamics Test for Closed Loop with Fixed Base

Table 6.16.: Results of the Forward Dynamics Test

Joint	$\ddot{q}_{SP}$ $[\frac{\mathrm{m}}{\mathrm{s}^2}]$	$\ddot{q}_{SDL}$ $[\frac{\mathrm{m}}{\mathrm{s}^2}]$
0	0	0
1	3.070515	3.0635261271
2	0	0
3	0	0
4	0	0
5	0	0
6	0	0
7	-3.149247	-3.1420780791

Table 6.17.: Results of the Inverse Dynamics Test

Joint	τ_{in} [Nm]	τ_{out} [Nm]
1	1	1
2	0	0
3	0	0
4	0	0
5	0	0
6	0	0
7	0	0

6.3.5. Test of Dynamics for Fixed Base – 2

Now the same functions **f_dyn_CL_fix** and **i_dyn_CL_fix** are tested with a different robot model, in order to validate the functions. The new model is built from the model used before. A second arm, similar to the already existing is added. Then, both end effectors are connected to perform a closed loop motion. The def-file of this model is displayed in Listing A.7. The model configuration is shown in Table 6.18.

Table 6.18.: Model Configuration of Dynamic Test

Variable	Value	Description
m	LBR3_JtoJ_light_CL2.def	Model
m.q[1]	0.7768443341 deg	joint 1 angle
m.q[2]	-63.8578805898 deg	joint 2 angle
m.q[4]	26.1951589399 deg	joint 4 angle
m.q[8]	-3.0500640798 deg	joint 8 angle
m.q[9]	64.0131671931 deg	joint 9 angle
m.q[10]	8.6962449759 deg	joint 10 angle
m.q[11]	-26.1951589399 deg	joint 11 angle
m.tau[2]	1 Nm	joint 1 torque

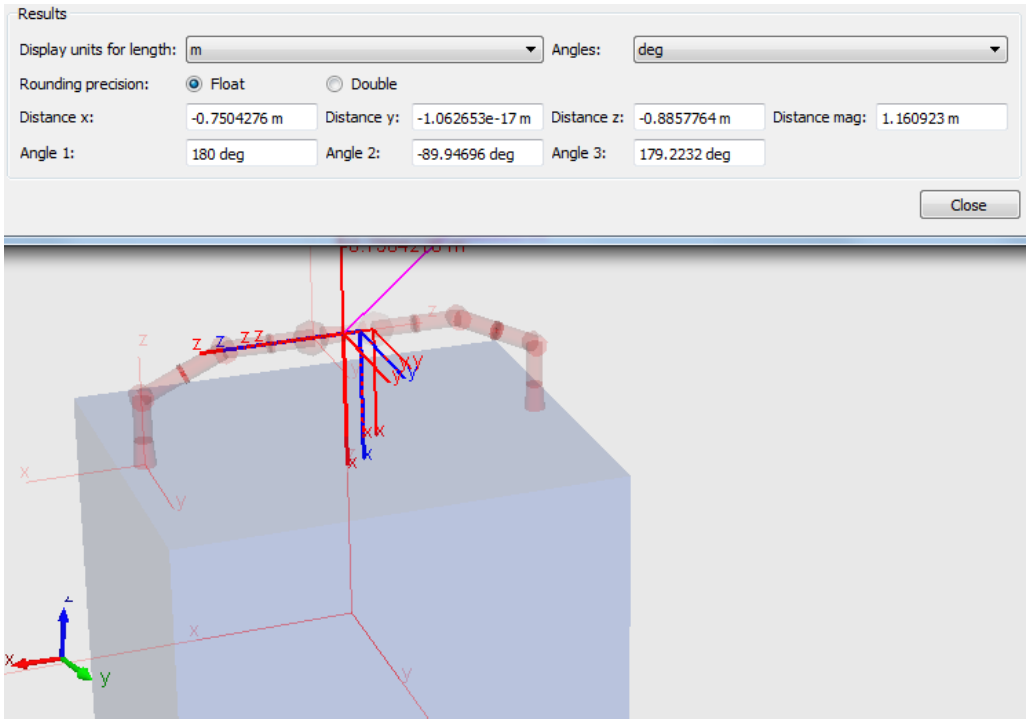
First the forward kinematics test is performed in order to validate the model representation. The results are shown in Figure 6.10 and Table 6.19.

Table 6.19.: Results of Forward Kinematics Test

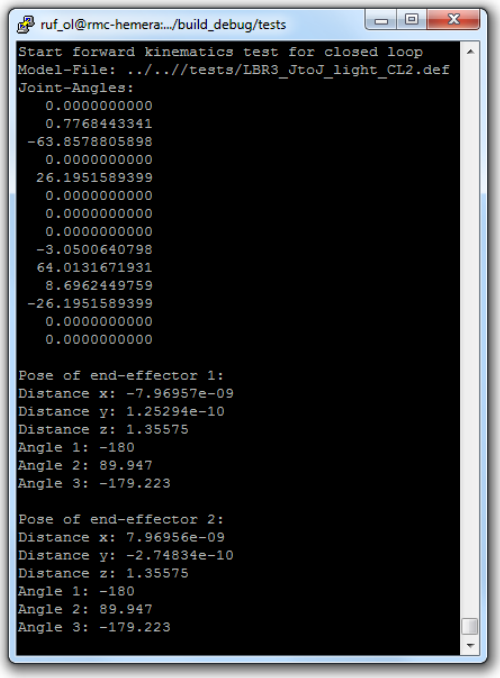
Variable	SIMPACK	SDL
Distance x	-7.9696e-09 m	-7.96957e-09 m
Distance y	1.2529e-10	1.25294e-10 m
Distance z	1.3558 m	1.35575 m
Angle 1	-180 deg	-180 deg
Angle 2	89.94696 deg	89.947 deg
Angle 3	-179.2232 deg	-179.223 deg

After the successful forward kinematics test with the new model, the tests of the dynamic functions are performed. For the beginning, only joint 2 is actuated by applying a torque. Unfortunately the forward dynamics function for fixed base robots with kinematic loops **f_dyn_CL_fix** throws an error because of occurring singularities. Nevertheless, the inverse dynamics function is tested by using the joint accelerations obtained with SIMPACK as the input of **i_dyn_CL_fix**. The result is displayed in Figure 6.11 and Table 6.20. The calculated torque is very accurate and corresponds with the input value.

At the point of writing the forward dynamics function was still not working for every robotic model. The error is due to singularities by inverting matrices, that are no square matrices. A function for building the pseudo-inverse is implemented in the



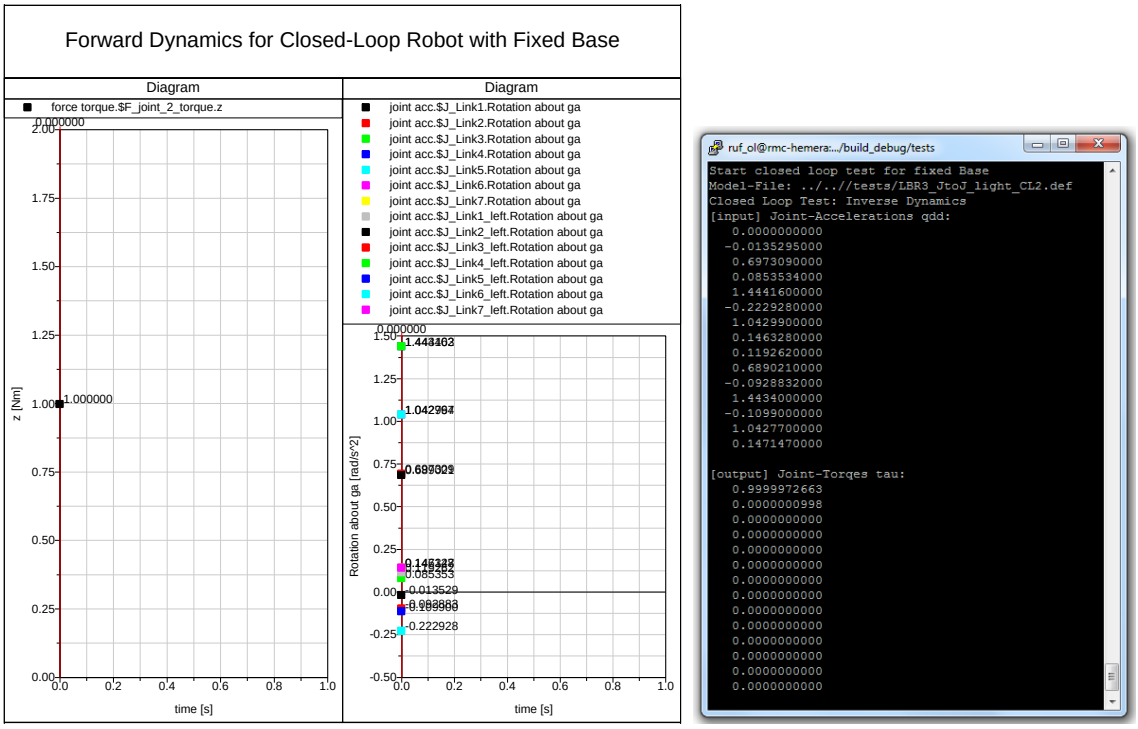
(a) SIMPACK



(b) SpaceDynamics Library

Figure 6.10.: Results of Forward Kinematics Test

SpaceDynamics Library and used for example if $\boldsymbol{S}$ is not a square matrix but it is not working for all systems.



(a) SIMPACK (b) SpaceDynamics Library

Figure 6.11.: Inverse Dynamics Test for Closed Loop

Joint	τ_{in} [Nm]	τ_{out} [Nm]
1	1	-0.9999972663
2	0	-0.0000000998
3	0	0
4	0	0
5	0	0
6	0	0
7	0	0
8	0	0
9	0	0
10	0	0
11	0	0
12	0	0
13	0	0
14	0	0

Table 6.20.: Results of the Inverse Dynamics Test

7. Results, Conclusion and Future Work

7.1. Results

With the results obtained by the tests described in the preceding chapter, the appropriate operation of the SpaceDynamics Library for kinematic trees is proved. The results calculated by the functions of the SpaceDynamics Library correspond with the results obtained by SIMPACK. The kinematic tree dynamic functions are therefore valid functions and can be used to simulate fixed base, free floating or free flying robotic systems. The functions for the dynamic modeling of robots with kinematic loops are however not working for all robotic systems, as shown in the preceding chapter. One main problem occurs when there are more actuators than degrees of freedom in the system. Then the system is over-determined and its not always possible to calculate the results. The problem can be solved by defining the joints as actuated and passive joints, however, this is normally not known beforehand.

Investing more time into this research could solve the problem and would lead to an algorithm that is able to handle all kind of systems. However there were some time consuming issues during this research, which made the goal unachievable in the given period of time. The first issue was the theory because it is very complex. It took a lot of time, in order to understand the individual basics and put them together to solve other problems. In order to implement these algorithms, the understanding of all functions and the operation of the SpaceDynamics Library was necessary. In the SpaceDynamics Library almost no comments existed in the code, so this task was even more time consuming. Furthermore the array representations are inconsistent, because some arrays start at index 0, others at index 1. Probably the most time consuming part was to identify and remove errors of different functions, that were implemented by others. It is very hard to understand what a function is doing if there are no comments or documentation, and it is even harder to find errors in those functions. On the other hand, the smallest issue of the research was to learn working with the simulation environment SIMPACK. It was rather easy to build and simulate models, however it takes a certain amount of time to get to know the program.

Nevertheless, the thesis reaches almost all goals. With the knowledge of the theory and the understanding of the SpaceDynamcis Library functions, only more time is needed to debug the code in order to validate the functions for robots with kinematic loops.

7.2. Conclusion

This report first described the motivation for the dynamic modeling of robots with kinematic loops. The dynamic modeling of robots with kinematic loops is for example needed for the simulation of the deorbiting of Envisat. Envisat is the largest Earth observing satellite of ESA and is not controllable anymore. Therefore, the deorbiting of Envisat would be an important step in actively reducing space debris with a service satellite. The report explained the terminology and definitions and gave a fundamental insight into the topic. Examples and applications of robotic systems with kinematic loops were given. Among the various applications, cooperative manipulators, humanoid robots, space robots and parallel robots were found to be the most important examples of robots with kinematic loops. The theory and equations of the dynamic modeling for robots with and without kinematic loops were explained. The theory and the equations also answer the research question of the thesis: How to compute the inverse and forward dynamics for a fixed base, free-flying or free-floating robotic system including kinematic and actuation redundancy with N arms and L closed loops. Two different methods of modeling kinematic loops were described in detail and compared with each other. The SpaceDyn library, which is used at the DLR German Aerospace Center was presented. It is a library of functions that calculates and simulates the kinematics and dynamics of a robot. In this scope the structure and the most important functions of the SpaceDynamics Library were described. After that, the implementation of the new algorithms for calculating the dynamics of robots with kinematic loops was shown. The tests and the results of the existing and newly implemented algorithms were presented and proved the SpaceDynamics Library to be valid to a certain extent. The goal of the research, to simulate the dynamics of a closed loop robotic system with the SpaceDynamics Library was reached for some robot models.

7.3. Future Work

Further work of this research includes solving the the forward dynamics problem for a robot with kinematic loops. Then the algorithms should be applied to a more realistic model, which is illustrated in Figure 7.1. This model is very close to the Envisat Case, where an object is grasped be two arms. It was already implemented in Listing A.7. As soon as the dynamics functions for kinematic loops are validated with this model, the next step is to control a similar robotic system. For the DLR it would be interesting to model and control the DEOS simulator with these algorithms. The DEOS simulator is simulating the grasping of a tumbling satellite with a manipulator of a service satellite for maintenance or deorbiting. In the simulator, a closed loop occurs over the simulator robots and the ground, as soon as the manipulator grasps the target satellite. Figure 7.2 shows the DEOS Simulator. On the left side is the tumbling target satellite, on the right side is the service satellite with the orange arm, which is pointing towards the target satellite. DEOS is is the first mission that aims on reducing space debris actively.

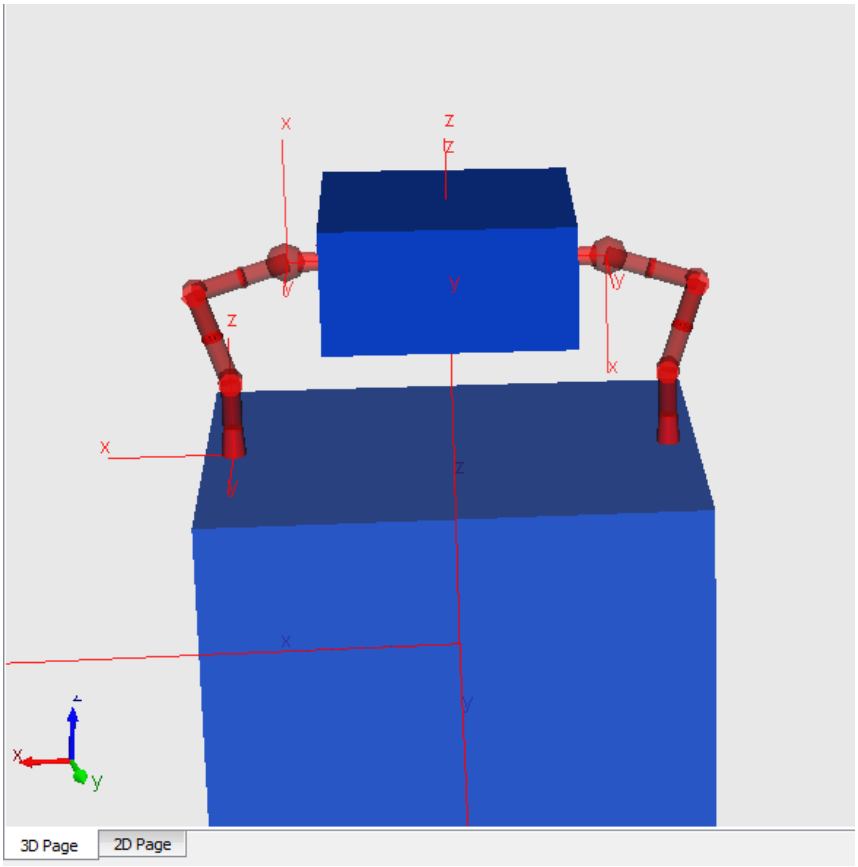


Figure 7.1.: Realistic Test Model

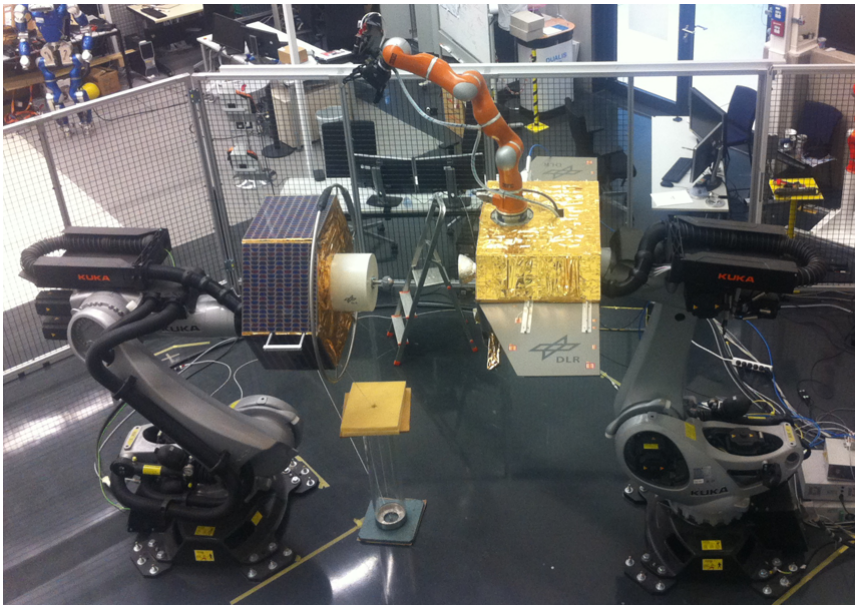


Figure 7.2.: DEOS Simulator

A. Listings

A.1. Tests

The following tests are implemented in spacedyn/tests/spacedyn_test.cpp

Listing A.1: Forward Kinematics Test

```
/**
 * Tests the forward kinematics for a model without closed loops
 *
 * Computes the pose of the end-effector
 *
 * @param output verbose output
 */
void test_forward_kinematics(bool output){
  if(output) std::cout << "Start forward kinematics test" << endl;

  // file
  string path("../../");
  string filename("LBR3_JtoJ_light.def");
  string modelfile = path + "/tests/" + filename;
  if(output) std::cout << "Model-File: " << modelfile << endl;

  // model
  MODEL m;
  // load model without output and without closed loops
  model(modelfile.c_str(), m ,false,false);
  model_init(m);

  // joint angles
  m.q[2] = deg2pi(90.0);
  m.q[3] = deg2pi(90.0);
  m.q[4] = deg2pi(90.0);

  // calculate spatial notation
  calc_SP(m);

  // calculate position (m.Pos_e) and orientation (m.ORI_e) of the end-effector
  f_kin_e(m,1);

  // convert orientation rotation matrix to roll, pitch and yaw angles
  Vector3 RPY_EE;
  R2rpy(m.ORI_e[1],RPY_EE);

  // convert rad joint angles to deg
  double* angles = new double[m.LINKNUM];
  matrix_cpy(m.LINKNUM,1,m.q,angles);
  for (int i=0;i<m.LINKNUM; i++){
    angles[i]=pi2deg(angles[i]);
  }

  // print results
  if(output) {
    std::cout << "Joint-Angles:" << endl;
    matrix_print(m.LINKNUM-1,1,angles);
    std::cout << "Pose of end-effector:" << endl;
    std::cout << "Distance x: " << m.POS_e[1][0] << endl;
```

```
51      std::cout << "Distance y: " << m.POS_e[1][1] << endl;
        std::cout << "Distance z: " << m.POS_e[1][2] << endl;
53      std::cout << "Angle 1: " << pi2deg(RPY_EE[0]) << endl;
        std::cout << "Angle 2: " << pi2deg(RPY_EE[1]) << endl;
55      std::cout << "Angle 3: " << pi2deg(RPY_EE[2]) << endl;
        std::cout << endl;
57    }

59    m.destruct_ee();
      m.destructor();
61  }
```

Listing A.2: Dynamics for Fixed Base Kinematic Tree System

```
1   /**
     * Tests the inverse and forward dynamics for a fixed base model without closed
        loops
3    *
     * Computes the joint accelerations qdd for given joint torques/forces m.tau and
5    * computes required joint torques/forces tau for required joint accelerations m.
        qdd
     *
7    * @param output verbose output
     */
9   void test_open_loop_fix(bool output){
      if(output) std::cout << "Start open loop test for fixed Base" << endl;
11
      // file
13    string path("../../");
      string filename("LBR3_JtoJ_light.def");
15    string modelfile = path + "/tests/" + filename;
      if(output) std::cout << "Model-File: " << modelfile << endl;
17
      // model
19    MODEL m;
      // load model without output and without closed loops
21    model(modelfile.c_str(), m ,false,false);
      model_init(m);
23
      // Gravity
25    double *gravity;
      gravity = vector_get(6);
27    vector_z(6,gravity);
      gravity[2]=-9.81;
29
      if(output) std::cout << "Open Loop Test: Forward Dynamics" << endl;
31    // input torques
      m.tau[2] = 1;
33    m.tau[4] = -1;
      m.q[2]=deg2pi(90.0);
35    m.q[3]=deg2pi(90.0);
      m.q[4]=deg2pi(90.0);
37    m.qd[2]=-1;

39    // output variable
      double* qdd;
41    qdd = vector_get(m.LINKNUM);
      vector_z(m.LINKNUM,qdd);
43
      // calculate spatial notation
45    calc_SP(m);

47    // calculate joint accelerations qdd
      f_dyn_fix(m,gravity,qdd);
49
      // print results
51    if(output) {
      std::cout << "[input] Joint-Torqes tau:" << endl;
53    matrix_print(m.LINKNUM,1,m.tau);
      std::cout << "[output] Joint-Accelerations qdd:" << endl;
```

```
55    matrix_print(m.LINKNUM,1,qdd);
      }
57
      if(output) std::cout << "Open Loop Test: Inverse Dynamics" << endl;
59    // input accelerations
      m.qdd = qdd;
61
      // output variable
63    double* tau;
      tau = vector_get(m.LINKNUM);
65    vector_z(m.LINKNUM,tau);

67    // calculate joint torques tau
      i_dyn_fix(m,gravity,tau);
69
      // print results
71    if(output) {
      std::cout << "[input] Joint-Accelerations qdd:" << endl;
73    matrix_print(m.LINKNUM,1,m.qdd);
      std::cout << "[output] Joint-Torqes tau:" << endl;
75    matrix_print(m.LINKNUM,1,tau);
      }
77
      m.destruct_ee();
79    m.destructor();
}
```

A.2. Functions

Listing A.3: Inverse Dynamics for Fixed Base Kinematic Tree System

```
/** Inverse Dynamics for a fixed base multi-body system
2  *
   * Calculates the required forces/torques to generate a given motion for a fixed
       base system.
4  * The algorithm implements the following equation directly and is therefore rather
   * slow. It has a computational complexity of O( n   )
6  *
   * Equation:
8  * tau = Hm * m.qdd + C
   *
10 * calc_SP must be called prior to this function.
   *
12 * The motion is given by
   *  - MODEL::qd
14 *  - MODEL::qdd
   *  and the kinematic situation:
16 *  - MODEL::Xup (thus indirectly MODEL::q)
   *  Additional inputs are the given forces on the end-effectors:
18 *  - MODEL::Fe
   *  - MODEL::Te
20 *
   * @param[in] m MODEL class
22 * @param[in] Gravity vector (6 x 1)
   * @param[out] tau joint_torques (LINKNUM-1 x 1)
24 *
   * @author { Oliver Ruf}
26 * @date 2014
   *
28 */
void i_dyn_fix_matrix_based(MODEL &m, double *Gravity, double *tau) {
30   // Variable initialization
     double *HH, *Hm, *CC, *Cm,*tmp;
32   HH = matrix_get( 6+m.LINKNUM-1, 6+m.LINKNUM-1 );
     Hm = matrix_get( m.LINKNUM-1, m.LINKNUM-1 );
```

```
  CC = vector_get(6+m.LINKNUM-1);
  Cm = vector_get(m.LINKNUM-1);
  tmp = vector_get(m.LINKNUM-1);

  // calculate the inertia matrix
  calc_hh(m,HH);
  // extract the manipulator inertia matrix
  matrix_ext_sub( 6+m.LINKNUM-1, 6+m.LINKNUM-1, 7, 6+m.LINKNUM-1, 7, 6+m.LINKNUM
    -1, HH, Hm );

  // calculate non-linear velocity dependent term
  calc_C(m, Gravity ,CC);
  // extract the manipulator term
  matrix_ext_sub(6+m.LINKNUM-1,1,7,6+m.LINKNUM-1,1,1, CC,Cm );

  // calculate inverse dynamics
  matrix_mult(m.LINKNUM-1,m.LINKNUM-1,1,Hm,m.qdd,tmp);
  matrix_add(m.LINKNUM-1,1,tmp,Cm,tau);

  delete[] HH;
  delete[] Hm;
  delete[] CC;
  delete[] Cm;
  delete[] tmp;
}
```

Listing A.4: Forward Dynamics for Fixed Base Kinematic Tree System

```
/** Forward Dynamics for a fixed base multi-body system
*
* Calculates the acceleration qdd for given forces/torqes tau for a fixed base
    system.
* The algorithm implements the following equation directly and is therefore rather
* slow. It has a computational complexity of O( n  )
*
* Equation:
* qdd = Hm_inv (m.tau-C)
*
* calc_SP must be called prior to this function.
*
* The motion is given by
*  - MODEL::tau
*  and the kinematic situation:
*  - MODEL::Xup (thus indirectly MODEL::q)
*  Additional inputs are the given forces on the end-effectors:
*  - MODEL::Fe
*  - MODEL::Te
*
* @param[in] m MODEL class
* @param[in] Gravity vector (6 x 1)
* @param[out] qdd joint accelerations (LINKNUM-1 x 1)
*
* @author { Oliver Ruf}
* @date 2014
*
*/
void f_dyn_fix_matrix_based(MODEL &m, double *Gravity, double *qdd) {
  // Variable initialization
  double *HH, *Hm, *Hm_inv, *CC, *Cm,*tmp;
  HH = matrix_get( 6+m.LINKNUM-1, 6+m.LINKNUM-1 );
  Hm = matrix_get( m.LINKNUM-1, m.LINKNUM-1 );
  Hm_inv = matrix_get( 6+m.LINKNUM-1, 6+m.LINKNUM-1 );
  CC = vector_get(6+m.LINKNUM-1);
  Cm = vector_get(m.LINKNUM-1);
  tmp = vector_get(m.LINKNUM-1);

  // calculate the inertia matrix
  calc_hh(m,HH);
  // extract the manipulator inertia matrix
```

```
41    matrix_ext_sub(  6+m.LINKNUM-1, 6+m.LINKNUM-1, 7, 6+m.LINKNUM-1, 7, 6+m.LINKNUM
        -1, HH, Hm );
      // invert Hm
43    matrix_inv(m.LINKNUM-1,Hm,Hm_inv);

45    // calculate non-linear velocity dependent term
      calc_C(m,Gravity,CC);
47    // extract the manipulator term
      matrix_ext_sub(6+m.LINKNUM-1,1,7,6+m.LINKNUM-1,1,1, CC,Cm );
49
      // calculate forward dynamics
51    matrix_sub(m.LINKNUM-1,1,m.tau,Cm,tmp);
      matrix_mult(m.LINKNUM-1,m.LINKNUM-1,1,Hm_inv,tmp,qdd);
53
      delete[] HH;
55    delete[] Hm;
      delete[] Hm_inv;
57    delete[] CC;
      delete[] Cm;
59    delete[] tmp;
}
```

A.3. Def-Files

Listing A.5: spacedyn/tests/LBR3_JtoJ_light.def

```
###Parameters_for_MODEL(__DO_NOT_CHANGE_THIS_ORDER!!___)
2
###LINK_NUMBER 8
4
###LINK_CONNECTIVITY
6 BB[ 0 1 2 3 4 5 6 ]

8 ###Joint_Type_[_0=rotational__1=prismatic]
J_type[ 0 0 0 0 0 0 0 ]
10
###EE
12 EE[ 0 0 0 0 0 0 1 ]

14 ###Relative_Coordinate_For_Link:Roll_Pitch_Yaw_Angle_of_Each_Bodies{x-y-z}_[deg]
rpy1[ 0 0 0 ]
16 rpy2[ 90 0 0 ]
rpy3[ -90 0 0 ]
18 rpy4[ -90 0 0 ]
rpy5[ 90 0 0 ]
20 rpy6[ 90 0 0 ]
rpy7[ -90 0 0 ]
22
###Vector_Of_Link_Length_JtoC_0_0_is_always_[_0_0_0_]
24 CtoJ_0_1[ 0.885 -0.012 0.98 ]

26 JtoC_1_1[ 0 0.01698 0.14087 ]
CtoJ_1_2[ 0 -0.01698 0.05913 ]
28 2.7082 2.71 2.5374 2.5053 1.3028 1.5686 0.1943
JtoC_2_2[ 0 0.1109 0.0141 ]
30 CtoJ_2_3[ 0 0.0891 -0.0141 ]

32 JtoC_3_3[ 0 -0.01628 0.13379 ]
CtoJ_3_4[ 0 0.01628 0.06621 ]
34
JtoC_4_4[ 0 -0.10538 0.01525 ]
36 CtoJ_4_5[ 0 -0.09462 -0.01525 ]

38 JtoC_5_5[ 0 0.01566 0.06489 ]
CtoJ_5_6[ 0 -0.01566 0.12511 ]
```

```
40
JtoC_6_6[ 0   0.00283   -0.00228 ]
CtoJ_6_7[ 0   -0.00283   0.00228 ]

JtoC_7_7[ 0   0   0.06031 ]


###Vector_To_End-Effector
CtoE_1[ 0.0   0.0   0.07569 ]

###Relative_Coordinate_For_End-Effector
rpyE_1[   0.0   0.0   0.0   ]

###Mass_Parameter
mass[ 724.10     2.7082    2.71     2.5374     2.5053     1.3028     1.5686     0.1943 ]

###Inertia_Matrix
###[_I11_I12_I13]
###[_I21_I22_I23]
###[_I31_I32_I33]

###Link0
I11= 277.21
I22= 410.07
I33= 420.26
I12= -11.33
I13= 21.85
I23= -3.8

###Link1
I11= -0.022632
I22= -0.022793
I33= 0.0049639
I12= 0.0
I13= 0.0
I23= 0.0

###Link2
I11= 0.024444
I22= 0.0052508
I33= 0.023995
I12= 0.0
I13= 0.0
I23= 0.0

###Link3
I11= -0.012993
I22= -0.01326
I33= 0.004697
I12= 0.0
I13= 0.0
I23= 0.0

###Link4
I11= 0.023167
I22= 0.0048331
I33= 0.022751
I12= 0.0
I13= 0.0
I23= 0.0

###Link5
I11= 0.023045
I22= 0.022408
I33= 0.0030151
I12= 0.0
I13= 0.0
I23= 0.0

###Link6
```

```
I11= 0.0033636
I22= 0.0029876
I33= 0.0029705
I12= 0.0
I13= 0.0
I23= 0.0

###Link7
I11= 7.93e-05
I22= 7.83e-05
I33= 0.0001203
I12= 0.0
I13= 0.0
I23= 0.0

###EOF 777
```

Listing A.6: spacedyn/tests/LBR3_JtoJ_light_CL.def

```
###Parameters_for_MODEL(__DO_NOT_CHANGE_THIS_ORDER!!___)

###LINK_NUMBER 8

###LINK_CONNECTIVITY
BB[ 0 1 2 3 4 5 6 ]

###Joint_Type_[_0=rotational__1=prismatic]
J_type[ 0 0 0 0 0 0 0 ]

###EE
EE[ 0 0 0 0 0 0 1 ]

###Lflag
CL[ -1 -1 -1 -1 -1 -1 0 ]

###Relative_Coordinate_For_Link:Roll_Pitch_Yaw_Angle_of_Each_Bodies{x-y-z}_[deg]
rpy1[ 0   0   0 ]
rpy2[ 90   0   0 ]
rpy3[ -90   0   0 ]
rpy4[ -90   0   0 ]
rpy5[ 90   0   0 ]
rpy6[ 90   0   0 ]
rpy7[ -90   0   0 ]

###Vector_Of_Link_Length_JtoC_0_0_is_always_[_0_0_0_]
CtoJ_0_1[ -0.885   -0.012   0.98 ]

JtoC_1_1[ 0   0.01698   0.14087 ]
CtoJ_1_2[ 0   -0.01698   0.05913 ]

JtoC_2_2[ 0   0.1109   0.0141 ]
CtoJ_2_3[ 0   0.0891   -0.0141 ]

JtoC_3_3[ 0   -0.01628   0.13379 ]
CtoJ_3_4[ 0   0.01628   0.06621 ]

JtoC_4_4[ 0   -0.10538   0.01525 ]
CtoJ_4_5[ 0   -0.09462   -0.01525 ]

JtoC_5_5[ 0   0.01566   0.06489 ]
CtoJ_5_6[ 0   -0.01566   0.12511 ]

JtoC_6_6[ 0   0.00283   -0.00228 ]
CtoJ_6_7[ 0   -0.00283   0.00228 ]

JtoC_7_7[ 0   0   0.06031 ]


###Vector_To_End-Effector
CtoE_1[ 0.0   0.0   0.07569 ]
```

```
###Relative_Coordinate_For_End-Effector
rpyE_1[   0.0   0.0   0.0   ]

###Mass_Parameter
mass[ 724.10     2.7082    2.71     2.5374     2.5053     1.3028     1.5686     0.1943 ]

###Inertia_Matrix
###[_I11_I12_I13]
###[_I21_I22_I23]
###[_I31_I32_I33]

###Link0
I11= 277.21
I22= 410.07
I33= 420.26
I12= -11.33
I13= 21.85
I23= -3.8

###Link1
I11= -0.022632
I22= -0.022793
I33= 0.0049639
I12= 0.0
I13= 0.0
I23= 0.0

###Link2
I11= 0.024444
I22= 0.0052508
I33= 0.023995
I12= 0.0
I13= 0.0
I23= 0.0

###Link3
I11= -0.012993
I22= -0.01326
I33= 0.004697
I12= 0.0
I13= 0.0
I23= 0.0

###Link4
I11= 0.023167
I22= 0.0048331
I33= 0.022751
I12= 0.0
I13= 0.0
I23= 0.0

###Link5
I11= 0.023045
I22= 0.022408
I33= 0.0030151
I12= 0.0
I13= 0.0
I23= 0.0

###Link6
I11= 0.0033636
I22= 0.0029876
I33= 0.0029705
I12= 0.0
I13= 0.0
I23= 0.0

###Link7
I11= 7.93e-05
```

```
I22= 7.83e-05
I33= 0.0001203
I12= 0.0
I13= 0.0
I23= 0.0

###EOF 777
```

Listing A.7: spacedyn/tests/LBR3_JtoJ_light_CL2.def

```
###Parameters_for_MODEL(__DO_NOT_CHANGE_THIS_ORDER!!___)

###LINK_NUMBER 15

###LINK_CONNECTIVITY
BB[ 0 1 2 3 4 5 6 0 8 9 10 11 12 13]

###Joint_Type_[_0=rotational__1=prismatic]
J_type[ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

###EE
EE[ 0 0 0 0 0 0 1 0 0 0 0 0 0 0 2]

###Lflag
CL[ -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 7 ]

###Relative_Coordinate_For_Link:Roll_Pitch_Yaw_Angle_of_Each_Bodies{x-y-z}_[deg]
rpy1[ 0 0 0 ]
rpy2[ 90 0 0 ]
rpy3[ -90 0 0 ]
rpy4[ -90 0 0 ]
rpy5[ 90 0 0 ]
rpy6[ 90 0 0 ]
rpy7[ -90 0 0 ]
rpy8[ 0 0 0 ]
rpy9[ 90 0 0 ]
rpy10[ -90 0 0 ]
rpy11[ -90 0 0 ]
rpy12[ 90 0 0 ]
rpy13[ 90 0 0 ]
rpy14[ -90 0 0 ]

###Vector_Of_Link_Length_JtoC_0_0_is_always_[_0_0_0_]
CtoJ_0_1[ -0.885 -0.012 0.98 ]

JtoC_1_1[ 0 0.01698 0.14087 ]
CtoJ_1_2[ 0 -0.01698 0.05913 ]

JtoC_2_2[ 0 0.1109 0.0141 ]
CtoJ_2_3[ 0 0.0891 -0.0141 ]

JtoC_3_3[ 0 -0.01628 0.13379 ]
CtoJ_3_4[ 0 0.01628 0.06621 ]

JtoC_4_4[ 0 -0.10538 0.01525 ]
CtoJ_4_5[ 0 -0.09462 -0.01525 ]

JtoC_5_5[ 0 0.01566 0.06489 ]
CtoJ_5_6[ 0 -0.01566 0.12511 ]

JtoC_6_6[ 0 0.00283 -0.00228 ]
CtoJ_6_7[ 0 -0.00283 0.00228 ]

JtoC_7_7[ 0 0 0.06031 ]

CtoJ_0_8[ 0.885 -0.012 0.98 ]

JtoC_8_8[ 0 0.01698 0.14087 ]
CtoJ_8_9[ 0 -0.01698 0.05913 ]

```

```
JtoC_9_9[ 0   0.1109   0.0141 ]
CtoJ_9_10[ 0   0.0891   -0.0141 ]

JtoC_10_10[ 0   -0.01628   0.13379 ]
CtoJ_10_11[ 0   0.01628   0.06621 ]

JtoC_11_11[ 0   -0.10538   0.01525 ]
CtoJ_11_12[ 0   -0.09462   -0.01525 ]

JtoC_12_12[ 0   0.01566   0.06489 ]
CtoJ_12_13[ 0   -0.01566   0.12511 ]

JtoC_13_13[ 0   0.00283   -0.00228 ]
CtoJ_13_14[ 0   -0.00283   0.00228 ]

JtoC_14_14[ 0   0   0.06031 ]

###Vector_To_End-Effector
CtoE_1[ 0.0   0.0   0.07569 ]
CtoE_2[ 0.0   0.0   0.07569 ]

###Relative_Coordinate_For_End-Effector
rpyE_1[   0.0   0.0   0.0   ]
rpyE_2[   0.0   -180.0   0.0   ]

###Mass_Parameter
mass[ 724.10     2.7082    2.71     2.5374     2.5053     1.3028     1.5686     0.1943
     2.7082    2.71     2.5374     2.5053     1.3028     1.5686     0.1943 ]

###Inertia_Matrix
###[_I11_I12_I13]
###[_I21_I22_I23]
###[_I31_I32_I33]

###Link0
I11= 277.21
I22= 410.07
I33= 420.26
I12= -11.33
I13= 21.85
I23= -3.8

###Link1
I11= -0.022632
I22= -0.022793
I33= 0.0049639
I12= 0.0
I13= 0.0
I23= 0.0

###Link2
I11= 0.024444
I22= 0.0052508
I33= 0.023995
I12= 0.0
I13= 0.0
I23= 0.0

###Link3
I11= -0.012993
I22= -0.01326
I33= 0.004697
I12= 0.0
I13= 0.0
I23= 0.0

###Link4
I11= 0.023167
I22= 0.0048331
I33= 0.022751
```

```
I12= 0.0
I13= 0.0
I23= 0.0

###Link5
I11= 0.023045
I22= 0.022408
I33= 0.0030151
I12= 0.0
I13= 0.0
I23= 0.0

###Link6
I11= 0.0033636
I22= 0.0029876
I33= 0.0029705
I12= 0.0
I13= 0.0
I23= 0.0

###Link7
I11= 7.93e-05
I22= 7.83e-05
I33= 0.0001203
I12= 0.0
I13= 0.0
I23= 0.0

###Link8
I11= -0.022632
I22= -0.022793
I33= 0.0049639
I12= 0.0
I13= 0.0
I23= 0.0

###Link9
I11= 0.024444
I22= 0.0052508
I33= 0.023995
I12= 0.0
I13= 0.0
I23= 0.0

###Link10
I11= -0.012993
I22= -0.01326
I33= 0.004697
I12= 0.0
I13= 0.0
I23= 0.0

###Link11
I11= 0.023167
I22= 0.0048331
I33= 0.022751
I12= 0.0
I13= 0.0
I23= 0.0

###Link12
I11= 0.023045
I22= 0.022408
I33= 0.0030151
I12= 0.0
I13= 0.0
I23= 0.0

###Link13
I11= 0.0033636
```

```
200 I22= 0.0029876
    I33= 0.0029705
202 I12= 0.0
    I13= 0.0
204 I23= 0.0

206 ###Link14
    I11= 7.93e-05
208 I22= 7.83e-05
    I33= 0.0001203
210 I12= 0.0
    I13= 0.0
212 I23= 0.0

214 ###EOF 777
```

Bibliography

[1] H. Klinkrad, *Space Debris: Models and Risk Analysis*. Springer, 2006.

[2] *IADC Space Debris Mitigation Guidelines*. IADC, 2007.

[3] H.-G. Reimerdes, "Mikrometeroiten und space debris," in *Handbuch der Raumfahrttechnik*, pp. 117–129, Hanser, 2011.

[4] J. Sommer, "Rendezvous und Docking," in *Handbuch der Raumfahrttechnik*, pp. 430–443, Hanser, 2011.

[5] G. Hirzinger, K. Landzettel, and C. Kaiser, "Neue Technologien und Robotik," in *Handbuch der Raumfahrttechnik*, pp. 597–619, Hanser, 2011.

[6] M. Mejia-Kaiser, "Esa's choice of futures: Envisat removal or first liability case," in *63rd International Astronautical Congress*, 2012.

[7] K. Waldron and J. Schmiedeler, "Kinematics," in *Springer Handbook of Robotics*, pp. 9–33, Springer Berlin Heidelberg, 2008.

[8] R. Featherstone and D. E. Orin, "Dynamics," in *Springer Handbook of Robotics* (B. Siciliano and O. Khatib, eds.), pp. 35–65, Springer Berlin Heidelberg, 2008.

[9] R. Featherstone, *Robot Dynamics Algorithms*. The Springer International Series in Engineering and Computer Science Series, Kluwer Academic Pub, 1987.

[10] K. Yoshida and B. Wilcox, "Space robots and systems," in *Springer Handbook of Robotics* (B. Siciliano and O. Khatib, eds.), pp. 1031–1063, Springer Berlin Heidelberg, 2008.

[11] S. Abiko, R. Lampariello, and G. Hirzinger, "A dynamic library for versatile modeling of free-flying and mobile robotic systems," in *10th ESA Workshop on Advanced Space Technologies for Robotics and Automation (ASTRA)*, (Noordwijk, The Netherlands), ESA/ESTEC, 11 2008.

[12] Y. Nakamura and K. Yamane, "Dynamics computation of structure-varying kinematic chains and its application to human figures," *Robotics and Automation, IEEE Transactions on*, vol. 16, no. 2, pp. 124–134, 2000.

[13] K. Lilly and C. Bonaventura, "A generalized formulation for simulation of space robot constrained motion," in *Robotics and Automation, 1995. Proceedings., 1995 IEEE International Conference on*, vol. 3, pp. 2835–2840 vol.3, 1995.

[14] S. A. A. Moosavian, R. Rastegari, and E. Papadopoulos, "Multiple impedance control for space free-flying robots," *Journal of Guidance, Control, and Dynamics*, vol. 28, pp. 939–947, 2014/01/28 2005.

[15] F. Caccavale and M. Uchiyama, "Cooperative manipulators," in *Springer Handbook of Robotics* (B. Siciliano and O. Khatib, eds.), pp. 701–718, Springer Berlin Heidelberg, 2008.

[16] J. Cortés and T. Siméon, "Motion planning algorithms for general closed-chain mechanisms," 2005.

[17] T. B. Wimböck, *Controllers for Compliant Two-Handed Dexterous Manipulation*. PhD thesis, TU Vienna, 2012.

[18] S. A. A. Moosavian and E. Papadopoulos, "Free-flying robots in space: an overview of dynamics modeling, planning and control," *Robotica*, vol. 25, pp. 537–547, 9 2007.

[19] S. A. A. Moosavian and E. Papadopoulos, "On the kinematics of multiple manipulator space free-flyers and their computation," *J. Field Robotics*, vol. 15, no. 4, pp. 207–216, 1998.

[20] E. Papadopoulos and S. A. A. Moosavian, "Dynamics and control of space free-flyers with multiple manipulators," *Advanced Robotics*, vol. 9, no. 6, pp. 603–624, 1994.

[21] S. Dubowsky and P. Boning, "Coordinated control of space robot teams for the on-orbit construction of large flexible space structures," *Advanced Robotics*, vol. 24, no. 3, pp. 303–323, 2010.

[22] Y. Ishijima, D. Tzeranis, and S. Dubowsky, "The on-orbit maneuvering of large space flexible structures by free-flying robots," *Proc. SAIRAS: 8 Int. Sympos. Artificial Intelligence Robot. Automat. Space*, pp. 5–8, 2005.

[23] D. King and C. Ower, "Orbital robotics spiral evolution for future exploration missions," in *Proceedings of the 8th International Symposium on Artificial Intelligence, Robotics and Automation in Space. Munich*, pp. 1–8, 2005.

[24] J.-P. Merlet and C. Gosselin, "Parallel mechanisms and robots," in *Springer Handbook of Robotics* (B. Siciliano and O. Khatib, eds.), pp. 269–285, Springer Berlin Heidelberg, 2008.

[25] M. Rose, "Automated generation of efficient real-time code for inverse dynamic parallel robot models," in *Robotic Systems for Handling and Assembly*, pp. 39–57, Springer, 2011.

[26] J. Y. S. Luh, M. W. Walker, and R. P. C. Paul, "On-line computational scheme for mechanical manipulators," *Journal of Dynamic Systems, Measurement, and Control*, vol. 102, pp. 69–76, 06 1980.

[27] J. Denavit and R. S. Hartenberg, "Kinematic Modelling for Robot Calibration," *Trans. ASME Journal of Applyed Mechanics*, vol. 22, pp. 215–221, 1955.

[28] S. Chiaverini, G. Oriolo, and I. Walker, "Kinematically redundant manipulators," in *Springer Handbook of Robotics* (B. Siciliano and O. Khatib, eds.), pp. 245–268, Springer Berlin Heidelberg, 2008.

[29] Y. Nakamura and M. Ghodoussi, "Dynamics computation of closed-link robot mechanisms with nonredundant and redundant actuators," *Robotics and Automation, IEEE Transactions on*, vol. 5, pp. 294–302, Jun 1989.

[30] J. J. Craig, *Introduction to robotics*, vol. 7. Addison-Wesley Reading, MA, 1989.

[31] Y. Nakamura, Y. Yokokohji, H. Hanafusa, and T. Yoshikawa, "Unified recursive formulation of kinematics and dynamics of robot manipulators," in *Japan-U.S.A. Symposium on Flexible Automation*, (Osaka, Japan), pp. 53–60, 1986.

[32] Y. Nakamura and T. Ropponen, "Actuation redundancy of a closed link manipulator," in *American Control Conference, 1990*, pp. 2294–2299, 1990.

[33] M. W. Walker and D. E. Orin, "Efficient dynamic computer simulation of robotic mechanisms," *Journal of Dynamic Systems, Measurement, and Control*, vol. 104, pp. 205–211, 09 1982.